НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Системне програмування

на тему: ЗАСОБИ АВТОМАТИЗАЦІЇ СКЛАДСЬКИХ ВИРОБНИЧИХ ПРОЦЕСІВ. ВИКОНАВЧА СИСТЕМА

Виконав: студент ІІ курсу, групи КВ-73мп
(шифр групи)

Кіндзер Мар'ян Степанович
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник ст. викладач Дробязко І.П.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент д.т.н., проф. Сімоненко В.П.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Рівень вищої освіти – другий (магістерський) за освітньо-науковою програмою

Спеціальність (спеціалізація) – 123 «Комп'ютерна інженерія» («Спеціалізовані комп'ютерні системи»)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.П. Тарасенко
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Кіндзеру Мар'яну Степановичу

1. Тема дисертації «Засоби автоматизації складських виробничих процесів. Виконавча система», науковий керівник дисертації Дробязко Ірина Павлівна, ст. викладач, затверджені наказом по університету від «___»_____ 2018 р. №_____
2. Термін подання студентом дисертації «7» грудня 2018 р.
3. Об'єкт дослідження: складські виробничі процеси.
4. Предмет дослідження: засоби автоматизації складських виробничих процесів.
5. Перелік завдань, які потрібно розробити:
   - описати призначення і область застосування;
   - навести технічні характеристики;
   - провести огляд існуючих рішень і обґрунтування вибору підходу до розроблення системи автоматизації складських виробничих процесів;
   - здійснити вибір і обґрунтування засобів реалізації системи автоматизації складських виробничих процесів;
   - розробити і описати виконавчу систему засобів реалізації складських виробничих процесів схему та алгоритм керуючої програми;
   - дослідити швидкість роботи та компактність виконавчої системи засобів реалізації складських виробничих процесів схему та алгоритм керуючої програми.
6. Орієнтовний перелік публікацій:
   - Тези доповіді "Комп'ютерний моніторинг і автоматизація складських виробничих процесів."
   - Тези доповіді "Операційна система реального часу для засобів автоматизації складських виробничих процесів"

7. Дата видачі завдання «5» вересня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.12.2017	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	30.12.2017	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	02.01.2018	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	19.03.2018	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	21.05.2018	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації; підготовка матеріалів доповіді на конференції ПМК-2018.	20.07.2018	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; робота над розділом з охорони праці	16.08.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	22.10.2018	

Студент М.С. Кіндзер

Науковий керівник дисертації І.П. Дробязко

РЕФЕРАТ

Актуальність теми. Сучасний складський виробничий процес – трудомістка і витратна за часом частина виробничого циклу. Зважаючи на людський фактор, час виконання операцій на складі може нераціонально зростати. Операції відбору, сортування та інвентаризації продукції здебільшого виконуються згідно паперових документів, що надаються робітникам, і не є автоматизованими. Це негативно впливає на час виконання завдань та ймовірність помилок. Зважаючи на велику кількість компаній, які мають повний виробничий цикл, впровадження нових сучасних технологій і автоматизація складських виробничих процесів та їх моніторинг дозволить підвищити ефективність роботи працівників складу та прискорити виконання складських операцій.

Об’єктом дослідження є складські виробничі процеси.

Предметом дослідження є засоби автоматизації складських виробничих процесів.

Мета роботи: підвищення ефективності і засобів автоматизації складських виробничих процесів за рахунок розширення загальної функціональності системи та використання сучасних програмно-апаратних засобів меншої вартості.

Наукова новизна системи полягає у можливість зручного налаштування кожного апаратного пристрою розробленою системою довільної кількості параметрів та визначених команд. Також система може вказувати працівнику на допущену ним помилку в комплектації замовлення, а також отримуючи дані з керуючої системи, візуально вказувати на довгі затримки у виконанні певного етапу робочого процесу.

Практична цінність:

1. В ході проведення аналізу досліджуваної області було визначено, що існуючі системи не надають статистичні показники виробничих процесів.
2. Розроблена зручна система налаштування програмно-апаратних

пристроїв системи.

Апробація роботи. Система для класифікація веб-сайтів на основі методів інтелектуального аналізу даних була представлена та обговорювалась на наукових конференціях магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 21-23 березня 2018 р.) та «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 14-16 листопада 2018 р.)**.**

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуто існуючі системи автоматизації складських виробничих процесів.

У другому розділі розглянуто засоби розробки керуючої системи моніторингу складських виробничих процесів.

У третьому розділі описано засоби, з допомогою яких було розроблено керуючу систему засобів автоматизації та її користувацький інтерфейс.

У четвертому розділі надано інформацію про результати експериментальних досліджень системи.

У висновках представлені результати проведеної роботи.

Робота представлена на 75 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: автоматизація, складські процеси, керуюча система, інтерфейс користувача.

ABSTRACT

Actuality of subject. Modern warehouse production process is a labor-consuming and time-consuming part of the production cycle. Given the human factor, the timing of operations in stock may not rationally increase. The operations of selection, sorting and inventory of products are mostly carried out in accordance with the paper documents provided to the workers, and are not automated. This negatively affects the timing of tasks and the probability of errors. Given the large number of companies with a full production cycle, the introduction of new modern technologies and the automation of warehouse production processes and their monitoring will increase the efficiency of staffing and speed up the performance of warehouse operations. The object of the study is classification of users of the social network Instagram on a gender basis.

The subject of the study warehouse production processes.

Methods of research - automation tools for warehouse production processes.

The purpose of the work: to increase of efficiency and means of automation of warehouse production processes by expanding the overall functionality of the system and using modern software and hardware of lesser value.

The scientific novelty of the system is the ability to conveniently configure each hardware device by a system of arbitrary number of parameters and specified commands. Also, the system may indicate to the employee the mistakes made in the order for it, as well as obtaining data from the control part, to indicate visually the long delays in the execution of a particular stage of the work process.

Work approbation. The main provisions and results of the work were presented and discussed at:

- XI scientific conference of masters and postgraduates "Applied Mathematics and Computering" PMK-2018-2 (Kiev, March 20-23, 2018);
- XI scientific conference of masters and postgraduates "Applied Mathematics and Computering" PMK-2018-2 (Kiev, November 14-17, 2018);

Publications. According to the results of the study 2 scientific works were published, including 1 article and 1 thesis of the conference.

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work

In the first section the theoretical information on the given topic is considered, as well as the analysis which allows to determine the main advantages and disadvantages of the existing methods of classification of images and text data.

The second section describes the modifications made to existing data analysis methods and describes how to use them.

The third section describes the measurement of the accuracy of the proposed methods and the description of the established classification system

The fourth section presents the experimental results of the developed system of classification of users on a gender basis

The conclusions are the results of the work.

The work is presented on 75 sheets, contains a link to the list of used literary sources.

Keywords: automation, warehouse processes, control system, user interface.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ERP – Enterprise Resource Planning

WMS – Warehouse Management system

CSS – Cascading Style Sheets – каскадні таблиці стилів;

ПЗ – Програмне Забезпечення;

UI – User Interface– користувацький інтерфейс;

UX – User Experience – досвід користування;

MVC – Model-view-Controller – архітектурний шаблон;

MVP – Model-View-Presenter – шаблон проектування;

MVVP – Model-View-ViewModel – шаблон проектування;

LLVM – Low Level Virtual Machine – низькорівнева віртуальна машина;

ARC – Automatic Reference Counting – автоматичний підрахунок посилань;

SDK – Software Development Kit – набір засобів розробки;

PDF – Portable Document Format – формат файлу;

URL – Uniform Resource Locator – стандартизована адреса ресурсу;

API – Application Programming Interface - прикладний програмний інтерфейс;

ORM – Object-Relational Mapping – технологія програмування;

BaaS – Backend as a Service platform – сервісна платформа;

ВСТУП

Сьогодні одним з найрозповсюдженіших способів керування робочим процесом в складських приміщеннях вітчизняних компаній є такий, де, здебільшого, всі операції, що мають бути виконані, описуються згідно паперової документації, що видається робітнику. Очевидно, що при використанні такого способу значно збільшується час обробки отриманих вказівок та їх виконання. Більш того, існує ймовірність некоректного завершення завдання.

Зважаючи на це, за останні два десятиліття було розроблено декілька принципів автоматизації складських виробничих процесів (Pick to light, Pick by voice), котрі мають на меті спростити робочий процес працівника шляхом надання покрокових візуальних або звукових вказівок. Проте дані технології мають декілька недоліків, наприклад таких, як: висока вартість імплементації в складське приміщення, відсутність можливості моніторингу ефективності роботи працівників тощо.

З огляду на вище вказане було вирішено розробити виконавчу систему автоматизації складського виробничого процесу, яка підтримуватиме розроблений протокол комунікації по визначеній шині даних з керуючою системою. Дана система повинна мати змогу бути розширеною як функціонально, так і апаратно, а також здійснювати коректний аналіз вхідних цифрових та аналогових даних.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ СИСТЕМИ А АВТОМАТИЗАЦІЇ СКЛАДСЬКИХ ВИРОБНИЧИХ ПРОЦЕСІВ

1.1 Аналіз систем автоматизації складських виробничих процесів

Завдяки стрімкому розвитоку технологій за останні десятиліття вдалося розробити багато систем автоматизації для промислових компаній для зменшення помилок у виробничому процесі. Це дозволило зміцнити промисловість, і компанії, що почали використовувати програмні засоби для автоматизації виробничого циклу, підвищили ефективність робочого процесу.

Засоби автоматизації складських виробничих процесів є складними системами, тому для формування вимог до них перш за все потрібен аналіз самих складських процесів.

Сучасний складський виробничий процес – трудомістка і витратна за часом частина виробничого циклу [1]. Найчастіше діяльність на складах складається з ряду операцій, які можуть бути класифіковані як вхідні процеси, процеси зберігання та вихідні процеси (рис. 1.1). До вхідних процесів відноситься приймання та розміщення на складі товарів. До вихідних процесів відноситься відбір, перевірка, упаковка та транспортування продуктів відповідно до замовлень клієнтів. З усіх операцій на складі, відбір та комплектація замовлень є найскладнішими та трудомісткими операціями. Зважаючи на людський фактор, час виконання вихідних процесів на складі може нераціонально зростати. Операції відбору, сортування та інвентаризації продукції здебільшого виконуються згідно паперових документів, що надаються робітникам, і не є автоматизованими. Це негативно впливає на час виконання завдань та ймовірність помилок. Впровадження нових сучасних технологій і засобів автоматизації дозволить підвищити ефективність роботи працівників складу та, тим самим прискорити виконання складських операцій.

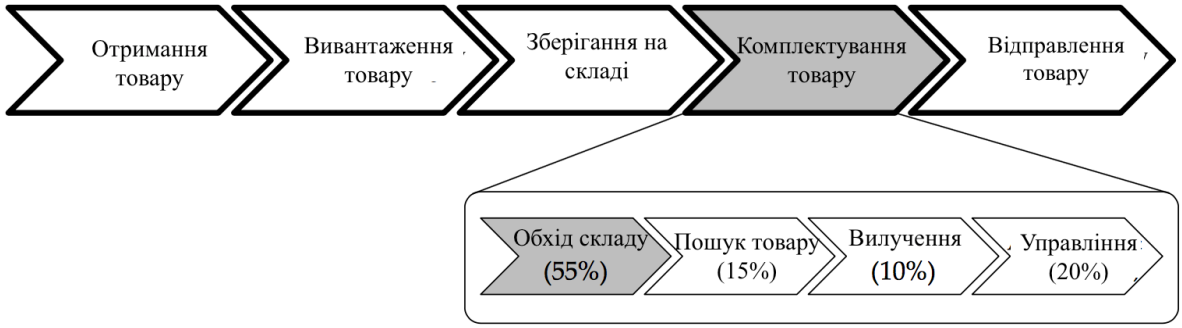

Рисунок 1.1 – Складські операції

Системою автоматизації складського виробничого процесу називається сукупність апаратно-програмних рішень, що автоматизують управління складськими процесами [2]. Найвідомішими з них є:

1. WMS (Warehouse Management system). Система управління складами WMS використовується переважно для управління складом [3]. Система відстежує рух кожного елемента товару, а потім отримує, збирає, упаковує та відправляє товар. Крім того, WMS, як правило, є автономними системами. Використання WMS може допомогти організації скоротити витрати на оплату праці, покращити точність інвентарю, підвищити гнучкість та оперативність, зменшити кількість помилок при виборі та доставці товарів. WMS пропонують оптимізацію процесу інвентаризації на основі інформації в реальному часі, що дозволяє компанії оперативно керувати такими процесами, як замовлення, відвантаження, надходження та будь-який рух товарів.
2. ERP (Enterprise Resource Planning). Програмне забезпечення ERP автоматизує бізнес-процеси в усіх відділах компанії, включаючи облік, ведення та обробку замовлень, закупівлю, управління запасами, управління складами, інтеграцію електронної комерції та управління відносинами з клієнтами [4].

Ці системи полегшують управління при щоденній організації планування робочого процесу і контроля використання наявних ресурсів.

Проте, незважаючи на вузькопрофільність, WMS є більш ефективнішою для керування складським виробничим процесом. Ця система може виконувати одну з найважливіших функцій – комплектацію замовлення, яка може виконуватись різними програмно-апаратними підходами, де покроково описаний кожний етап виконання робочого процесу працівникам.

Є декілька видів коплектації замовлення, в залежності від яких і обираються підходи до їх реалізації [5]:

1. Зонова комплектація. Ця комплектація передбачає закріплення окремої зони комплектації за окремим працівником. Таким чином, кожен працівник збирає всі позиції замовлення з зони зберігання, за якою він закріплений.
2. Комплектація по партіям. Працівник одночасно працює над кількома замовленнями. Спочатку він збирає товари одного типу. Далі вони відправляються на місце комплектації, де їх розподіляють за конкретними замовленнями.
3. Хвильова комплектація. Суть комплектації полягає в тому, що замовлення не збираються окремо, а об'єднуються в так звану "хвилю" комплектації. Але для цього потрібно попередньо розмістити товари в різні зони складу. Замовлення розділяються на частини, які відповідають зонам комплектації складу. Далі із сформованих частин замовлення збирається в цілому. Це дозволяє істотно оптимізувати весь процес, адже комплектація замовлень прискорюється за рахунок зменшення кількості переміщень між місцями відбору.

Системи автоматизації складських процесів, перш за все є інструментом, придбаним та використовуваним підприємствами для задоволення вимог замовника тоді, коли інвентаризація та робоче навантаження більші, ніж при виконанні операцій вручну. Тому основною

причиною придбання та встановлення таких систем є, як правило, бажання підтримувати зростання якості виконуваних замовлень.

Системи автоматизації складського процесу містять розгорнуту базу даних, основною функцією якої є забезпечення підтримки складських операцій. Вона містить об’єкти, які описують різні елементи виробничого процесу на складі такі як, покрокові етапи збірок різних пристроїв, які виробляє компанія.

Основними функціями таких систем є [6]:

1. Планування. Завершення щоденного плану роботи, вибір робочого навантаження, тобто замовлень, що повинні бути оброблені за день або зміну.
2. Організація робочого процесу. Дані системи повинні забезпечувати персонал документацією щодо замовлень, тобто що саме повинен виконати працівник і в якій кількості. Також важливою функцією організації робочого процесу є подання вказівок щодо розташування елементів замовлення на виділені їм місця.
3. Простота. Система повинна бути зрозумілою для працівників на складі для уникнення затримок в процесі її використання.
4. Навчання. Система автоматизації складського процесу повинна давати вказівки, як коректно виконувати дії на кожному етапі робочого процесу, щоб в подальшому працівник пришвидшував виконання поставленого завдання.
5. Контроль. Забезпечення моніторингу впродовж дня, що дає можливість своєчасно реагувати на проблеми та подавати дані для аналізу ефективності працівників.
6. Гнучкість. Система повинна мати змогу бути налаштованою згідно потреб різних компаній, які її встановили.

Системи автоматизацій складськими приміщеннями підтримують штатний склад працівників у виконанні процесів, необхідних для обробки всіх головних і багатьох незначних складських завдань, таких як прийом, перевірка та прийняття, збір, упаковка та складання замовлень. Вони також допомагають керувати та перевіряти кожен крок, зберігати та записувати весь рух товару.

Зважаючи на зростаючий технічний прогрес і потреби у підвищенні ефективності робочого процесу виникають нові підходи до автоматизації складських виробничих процесів.

1.2 Порівняльний аналіз підходів до комплектації замовлень в системах автоматизації складських виробничих процесів

Сьогодні відомі різні підходи до комплектації замовлень в системах автоматизації складських виробничих процесів. Розглянемо найвживаніші з них.

Паперова документація

Паперова документація створюється для ознайомлення працівника з деталями виконання робочого процесу [7]. Вона містить список відбору товарів, які підлягають відбору зі складу і використовується для виконання замовлень. Як правило, цей список надає інформацію про відвантаження товару, його розташування на складі та допоміжну інформацію про нього.

Основними складовими паперового способу комплектації є порядковий номер складового елемента комплектації та його локація. Найчастіше у такій системі відсутній опис робочого процесу, тому виникають додаткові витрати часу для ознайомлення з його специфікаціями.

Це найпримітивніший підхід для системи автоматизації, але завдяки своїй низькій собівартості він є одним з найрозповсюджених. Можна виділити наступні недоліки даного підходу:

1. Відсутність опису робочого процесу;

2. Систематичне накопичення або заміна документації при внесенні зміни;
3. Можлива похибка у виборі складового елемента замовлення на етапах збору замовлення;
4. Відсутність візуальної сигналізації щодо місця розташування потрібного елемента товару;
5. Відсутність моніторингу роботи працівників.

Зважаючи на вище зазначені недоліки стає зрозумілим, що при наявності сучасних технологій даний підхід для створення системи автоматизації складського виробничого процесу, окрім економічної складової, є неефективним та застарілим.

Pick to light

Pick to light - це підхід до комплектації замовлень, який використовує буквено-цифрові дисплеї та кнопки для керування вибором та обробкою елементів товару (рис 1.2) [8].

Рисунок 1.2 – Система Pick to light

У типовій системі Pick to light працівник сканує штрих-код, прикріплений до блоку з елементами замовлення, який є багаторазовим контейнером для їх зберігання. Буквено-цифрові дисплеї підсвічуються, щоб керувати працівником у місці зберігання елементів товару та вказувати кількість елементів, що необхідні для виконання замовлення. Оператор розміщує елементи в контейнері та підтверджує виконання операції, натискаючи відповідну кнопку на дисплеї. Екран дисплея продовжує світитись у робочій зоні оператора, направляючи його до наступної складової замовлення.

Також пристрої цієї системи дозволяють оператору за допомогою додаткових кнопок записувати варіанти кількості та інших основних даних елемента. Деякі дисплеї також дозволяють вказувати номери замовлень, наступну зону замовлення, та спеціальні інструкції для операторів.

Змінюючи напрямок потоку, система може стати системою Put to light, тобто реалізувати зворотний вибір, коли товари розміщуються в контейнери, згруповані за замовленням клієнта.

Організація процесу з зоновою комплектацією, яку частково реалізує Put to light, допомагає мінімізувати зайві переміщення на складі. Це в свою чергу дозволяє знизити час виконання формування комплекту.

Модулі відображення, як правило, підключаються до шини, яка постачає дані. Кожен модуль працює незалежно один від одного, що знижує ймовірність збою системи.

Деякі системи Pick to light можуть бути встановлені на мобільних телефонах, що дозволяє операторам виконувати кілька замовлень за однин прохід по складу. Системи Pick to light оснащені спеціалізованим програмним забезпеченням, яке може інтегруватися з програмним забезпеченням управління процесу постачання товару та системою управління складом (WMS). Допоміжне програмне забезпечення для Pick to light дає змогу змінювати розміри робочих зон, у відповідності до обсягу замовлень.

Переваги систем Pick to light полягають у підвищенні продуктивності та точності, зниженні витрат на оплату праці та більш ефективне ведення обліку. Зменшується час на визначення місця знаходження та читання паперових записів, що підвищує ефективність процесу.

Недоліком системи Pick to light є те, що найчастіше у ній відсутні програмні засоби для відображення етапів робочого процесу на моніторах комп'ютера. Через це інформація про комплектацію знаходиться на дисплеях, але вона є неповною, оскільки малі за розміром дисплеї не дозволяють розмістити всю інформацію, як наприклад докладні вказівки, щодо подальших дії з елементом, отриманим на певному етапі комплектації замовлення.

Також, як правило, дані системи не забезпечують моніторинг виконання роботи працівником.

Радіочастотна ідентифікація

Загальний підхід до побудови систем автомазації складського виробничого процесу з використанням методу радіочастотної ідентифікації (рис. 1.3) полягає в бездротовій передачі ідентифікатора елемента замовлення у вигляді унікального серійного номера продукту [9]. Це покращує точність даних, що зберігаються в системі, забезпечуючи негайне введення даних, зменшує ризик неправильної ідентифікації запасів, а також кількість необхідних у системі для роботи документів і прискорює отримання даних від неї.

Місця розташування елементу товару ідентифікуються штрих-кодом на кожному контейнері, де цей елемент знаходиться. Інтерактивна система сканування дозволяє працівнику отримувати дані про місцезнаходження товару з бази даних. Це дає точне уявлення про місця розташування елементів товару, виключивши оформлення документів вручну та їх подальше введення у базу даних.

RF сканери використовуються для читання штрих-кодів і передаванні інформації про елементи замовлення, їхню кількість місцезнаходження.

Коли відбувається процес збору замовлення, розроблена комп'ютерна програма передає сканеру необхідну інформацію. На екрані сканера відображаються дані про замовлення, включаючи його місце зберігання, опис, штрих-код та кількість для кожного складового елемента.

Рисунок 1.3 – Радіочастотна ідентифікація

Недоліком даної системи є відсутність сигналізування місця розташування елементів замовлення на кожному етапі виконання робочого процесу.

Pick by voice

Підхід Pick by voice (рис 1.4) пропонує використання голосу та відповідне програмне забезпечення для розпізнавання мови. Запропонований в кінці 1990-х років, за останній час його використання зростає з розвитком технології розпізнавання мови та зменшенням витрат для такого програмного забезпечення, а також собівартості мобільних комп'ютерів, на яких воно працює [10].

Така система передбачає гарнітуру, яку носять працівники складу. Вона підключається до мережі, звідки надходять вказівки працівнику щодо знаходження елемента комплектування замовлення, а також його опис.

Працівники підтверджують свої завдання голосовими командами а також кодами, надрукованими на місцях збереження елементів замовлення. Програмне забезпечення для розпізнавання мовлення певним чином реагує на відповіді і забезпечує перехід до наступного етапу роботи [8].

Рисунок 1.4 – Система Pick by voice

Найчастіше Pick by voice використовується замість паперових або мобільних комп'ютерних систем, які вимагають від працівників читання інструкцій та сканування штрих-кодів або введення ключових даних для підтвердження їх завдань. Незважаючи на те, що спочатку дана система використовувалась тільки для збирання замовлень, тепер всі складські функції, такі як прийом, розташування, збирання та відправлення, можуть бути скоординовані голосовими системами.

Реалізація голосових систем на складі надає такі переваги:

1. Збільшення точності комплектування замовлення;
2. Збільшення точності розміщення інвентарю;
3. Збільшення продуктивності персоналу;
4. Зниження часу навчання для нових працівників;
5. Зниження вартості друку та розповсюдження паперової документації.

Незважаючи на значні переваги, система Pick by voice має декілька недоліків. Найважливішим з них є висока вартість, адже не кожне підприємство може витрачати додаткові кошти на допоміжне обладнання.

Другим недоліком є відсутність візуального сигналізування місцезнаходження елемента збору замовлення. Не менш важливим недоліком є недосконаліть розпізнавання мови, що може призвести до неправильного комплектування замовлення.

Augmented reality picking

Підхід Augmented reality picking (ARP) використовує технологію розширеної реальності, поєднуючи в собі різні зорові та голосові рішення, що забезпечують швидкий огляд елементів замовлення на складі [11].

Комплектування замовлення методом розширеної реальності використовує інтелектуальні окуляри для об'єднання віртуальних зображень та інформації з навколишнім середовищем оператора. Використовуючи окуляри, оператор виконує надані команди та сканує штрих-коди продукту, що знаходяться всередині дисплея окулярів. Комбінація реальної та віртуальної інформації забезпечує більшу швидкість та точність, ніж попередні підходи.

Працівник-оператор за допомогою окулярів доповненої реальності завантажує партію замовлень, що підлягають комплектації, та посилає голосову команду через вбудований мікрофон для отримання наступної локації елемента замовлення. При переході до вказаного місця, працівник сканує штрих-код та отримується підтвердження, що він знаходиться в потрібному місці.

Вбудована система зору розпізнає продукт і перевіряє правильність вибору елемента. В кінці комплектації замовлення система оновлює статус замовлення. Оператор виконує попередні дії до завершення партії замовлень.

Перевагами ARP є:

1. Об'ємний графічний інтерфейс користувача. Повнокольорові дисплеї з високою роздільною здатністю забезпечують візуально еквівалентне зображення на дисплеї окулярів доповненої

реальності. Дисплеї також можуть відображати багато інформації без необхідності додаткового портативного пристрою;

2. Голосовий контроль обробки з вбудованими динаміками, мікрофоном і алгоритмом розпізнавання голосу;
3. Сканування штрих-коду та захоплення зображень. Завдяки вбудованій камері та здатності оброблювати відео потоки, сканування штрих-кодів є невід'ємною частиною технології ARP.
4. Глобальне позиціонування - вбудована система GPS-навігації з навігацією оператором та позиціонування в межах складу;
5. Керування жестами. Алгоритм розпізнавання жестів в окулярах доповненої реальності дозволяє розпізнати рухи голови або тіла, які можуть слугувати командами системи.

Незважаючи на цілу низку переваг дана система продовжує перебувати на стадії розробки і тестової експлуатації, адже принцип доповненої реальності відносно новий, тому вимагає багатьох допрацювань.

Також описаний підхід є найдорожчим, зокрема для вітчизняних компаній, що є на сьогоднішній час вагомим недоліком при виборі підходів до автоматизації складських процесів.

1.3 Обґрунтування теми магістерської дисертації та постановка задачі

Для багатьох компаній актуальною проблемою є забезпечення ефективного налагодження та прискорення виконання виробничого циклу. Суттєве місце в цьому циклі відводиться засобам автоматизації складських процесів, які забезпечують прийняття, розміщення, накопичення та тимчасове зберігання запасів матеріальних ресурсів, незавершеного виробництва та готової продукції.

Зважаючи на розвиток технічного прогресу, сучасні компанії намагаються досягти збільшення прибутків шляхом застосування нових технологій, апаратних і програмних засобів для автоматизації процесів

виробництва. Оскільки склади є однією з ланок виробництва, їх автоматизація є важливим елементом для підвищення ефективності.

Зважаючи на ряд переваг підходу Pick to light, в основі якого лежить візуальна сигналізація місцерозташування об'єкта, у якості базової концепції розроблюваних засобів автоматизації складських виробничих процесів пропонується використовувати саме його.

Завданням магістерської роботи є побудова виконавчої системи засобів автоматизації складських виробничих процесів. Для цього потрібно вирішити наступні задачі: визначити сучасні способи і засоби побудови виконавчої системи, що дозволять оптимізувати робочий процес, зменшити кількість помилок, які можуть виникнути при комплектації замовлення, а також зменшити час виконання комплектації замовлень.

Висновки до розділу 1

Було розглянуто основні типи систем автоматизації складських процесів та визначено їх переваги та недоліки. Однією з основних функцій цих систем є комплектування замовлень, тому було розроблено багато програмно-апаратних підходів, що допомагають швидко інтегровувати штатний склад працівників у робочий процес.

На основі аналізу сучасних проблем виробництва і зокрема складських процесів, сучасного стану їх автоматизації, доведено актуальність теми і доцільність розробки засобів автоматизації складських виробничих процесів за допомогою сучасних технологій і технічних засобів, що дозволить підвищити ефективність роботи складу і виробництва в цілому.

Запропоновано для побудови таких засобів використовувати розглянутими вище принципи Pick to light, доповнивши їх засобами моніторингу процесів на складі. Також сформульовано основні задачі, які мають бути вирішені при розробці засобів.

2. ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ АВТОМАТИЗАЦІЇ СКЛАДСЬКИХ ВИРОБНИЧИХ ПРОЦЕСІВ

Виходячи з основних завдань системи автоматизації складських виробничих процесів її можна поділити як мінімум на дві частини, а саме виконавчу та керуючу (рис 2.1). Керуюча частина являє собою програмне рішення. Це може бути комп'ютерний, мобільний або веб додатки, що можуть бути реалізовані різними сучасними програмними рішеннями.

Виконавча система повинна мати сукупність апаратних пристроїв (мікроконтролерів), у відповідності до етапів виробничого циклу, та забезпечувати моніторинг та візуалізацію процесу комплектації замовлень, які надходять до складу. Також вона має реалізовувати інтерфейси взаємодії цих пристроїв з керуючою системою для отримання інструкцій щодо комплектації та працівником для налаштування самого пристрою.

В магістерській дисертації розглядається реалізація виконавчої системи засобів автоматизації складських виробничих процесів.

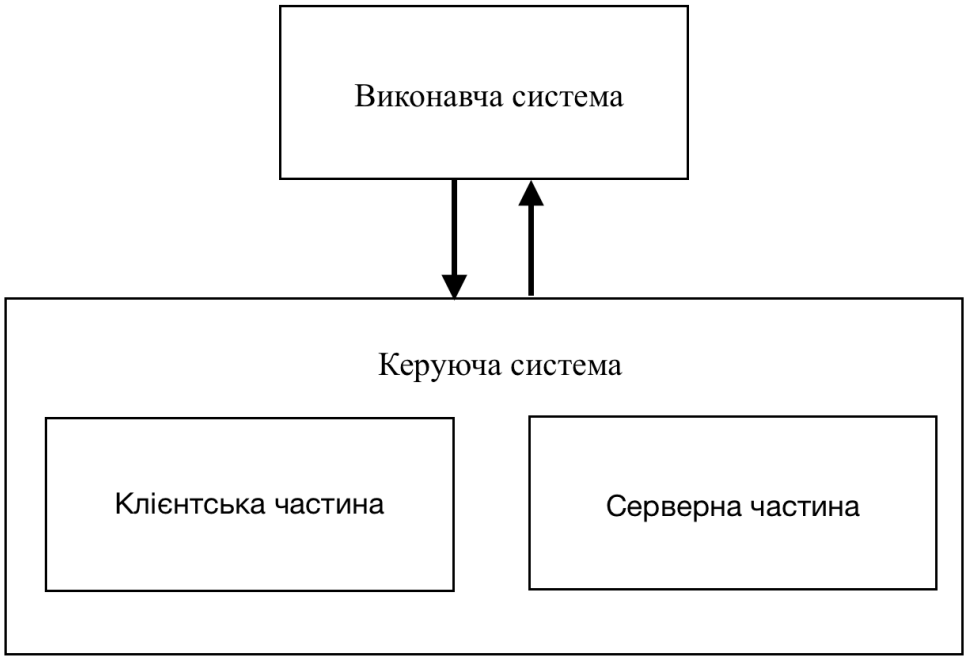

Рисунок 2.1 – Складові системи автоматизації

Для реалізації виконавчої системи необхідно визначити, які саме апаратні та програмні засоби відповідатимуть її задачам. Також необхідно, щоб сукупність цих засобів забезпечувала невисоку собівартість системи.

2.1 Мікроконтролери

Основою виконавчої системи є мікроконтролер. Мікроконтролером називається компактна інтегральна схема, призначена для управління певною операцією або операціями вбудованої системи. Основною складовою мікроконтролера є ядро процесора, на якому працює мікроконтролер. Ядром мікроконтролера, який є складовою виконавчої частини засобів автоматизації складських виробничих процесів вирішено обрати процесор Cortex ARM M0.

Процесор ARM Cortex M0 є одним з перших представників процесорів сімейства Cortex компанії ARM і націлений на ринок 32-бітних мікроконтролерів [12]. Даний процесор, незважаючи на невелику кількість логічних вентилів, необхідних для його реалізації, має велику продуктивність. Процесор Cortex M0 задовольняє різним вимогам ринку 32-бітних процесорів для вбудованих систем, пропонуючи:

1. Більшу продуктивність. Процесор дозволяє виконувати більший об'єм обчислень без необхідності збільшення частоти або споживчої потужності;
2. Низьке енерговикористання. Cortex M0 забезпечує більший час автономної роботи, що особливо критично для мікроконтролерів;
3. Покращений детермінізм. Cortex M0 гарантує, що перехід до обслуговування критичних задач і переривань буде здійснюватися за мінімально можливим і, головне, точно певним часом;
4. Збільшену щільність коду. Процесор дозволяє розмістити потрібний код навіть у пам'яті невеликого обсягу;

5. Простоту використання. Забезпечується легкість програмування та відтворення для зростаючого числа користувачів, що переходять з 8- і 16-бітових платформ на 32-бітну;
6. Низьку вартість. Вартість 32-бітних систем наближується до вартості класичних 8/16-бітових пристроїв і 32-бітові мікроконтролери початкового рівня коштують менше одного долару США;
7. Великий вибір засобів розробки – від недорогих або безкоштовних компіляторів до розроблених пакетів від різних виробників засобів розробки.

Ще одним з напрямків зниження собівартості є збільшення об'ємів коду, що використовується повторно в різних виробах. Оскільки мікроконтролери з процесорним ядром Cortex-M0 розраховані на програмування з використанням високорівневих мов, зокрема мови С, і мають встановлену архітектуру, це значно спрощує перенесення та повторне використання програм, зменшуючи тим самим час розробки та витрати на тестування.

Процесор Cortex-M0 представляє собою 32-бітний микропроцессор. Він має 32-бітну шину даних, 32-бітний банк регістрів та 32-бітні інтерфейси пам'яті.

Процессор виконаний по Гарвардській архітектурі, тобто має роздільні шини команд и даних. Це дозволяє здійснити вибірку команд одночасно з зверненням до даних. У результаті підвищується продуктивність процесора, оскільки операції доступу до даних ніяк не впливають на конвеєр команд. Це дозволило реалізувати в процесорі Cortex-M0 кілька шинних інтерфейсів, кожен з яких оптимізований для виконання певних функцій і, в той же час, може використовуватися одночасно з іншими інтерфейсами. При цьому шини команд і даних розділяють одне і те ж адресне місце (єдину пам'ять).

Для підтримки складних додатків, що вимагають більш розвиненої системи пам'яті, в процесорі Cortex-M0 передбачено додатковий модуль захисту пам'яті Memory Protection Unit (MPU) і можливість використання зовнішньої кеш-пам'яті. Підтримуються системи пам'яті, що використовують як прямий, так і зворотній порядок байтів.

У складі процесора Cortex-M0 є контроллер вкладених векторних переривань Nested Vectored Interrupt Controller (NVIC). Він тісно пов'язаний з ядром процесора і виконує наступні функції:

1. Підтримка вкладених переривань;
2. Підтримка векторних переривань;
3. Підтримка динамічної зміни приорітетів;
4. Зменшення затримок обробки переривання;
5. Маскування переривань.

В процесорі Cortex-M0 використовується фіксований розподіл адресного простору. Це дозволяє звернутися до вбудованих периферійних пристроїв, таких як контролер переривань, за допомогою звичайних команд доступу до пам'яті. Це означає, що більшість системних функцій можна використовувати безпосередньо з програм, написаних мовою програмування С. Визначена карта пам'яти також забезпечує надзвичайно високу швидкість роботи процесора і полегшує його інтеграцію в системі на кристалі System on a Chip (SoC).

Мікроконтролер на ядрі Cortex-M0 має внутрішню шинну інфраструктуру, яка оптимізирована для використання пам'яті, розподіленої відповідно до рис. 2.2. Крім того, конструкція процесора дозволяє задіяти зазначені області в різний спосіб. Так, пам'ять даних може бути розміщена в секції коду, а код програми може запускатися з секції зовнішнього ОЗУ.

Рисунок 2.2 – Структура розподілу пам'яті мікроконтролера Cortex M0

В системних областях пам'яті розташовані контролери переривань і компонентів відладки. Розміщення цих периферійних пристроїв за фіксованими адресами значно спрощує передачу прикладних програм між мікроконтролерами різних виробників.

2.2 Протоколи зв'язку

Для забезпечення зв’язку з керуючою системо, користувачем, а також з окремою енергонезалежною пам'яттю необхідно реалізувати протоколи деяких шин даних. Для комунікації між апаратним пристроєм та користувачем доцільно використовувати шину даних UART, оскільки існує багато програмного забезпечення, що відображає дані, отримані з шини даних.

Реалізація зв’язку мікроконтролера з енергонезалежною пам'яттю виконується шиною даних SPI. Обрана саме ця шина даних, оскільки основною метою її створення була необхідність взаємодії між процесором та енергонезалежною пам’яттю.

Взаємозв’язок виконавчої та керуючої систем забезпечує протокол CAN. Однією з його особливостей, якими був зумовлений вибір цієї шини даних, є принцип “арбітражного суду”, що дозволяє зручно керувати пристроями, якщо надати їм коректні адреси на цій шині даних.

UART

Протокол UART (Universal asynchronous receiver/transmitter) – це фізичний протокол прийому та передачі даних. У комунікації UART два пристрої, в яких є UART взаємодіють безпосередньо один з одним [13]. Передаючий пристрій перетворює паралельні дані з керуючого пристрою, такого як процесор у серійну форму, передає його послідовно в приймаючий пристрій, який потім перетворює серійні дані назад на паралельні дані (рис 2.3). Для передачі даних між двома UART потрібні лише два дроти.

UART передає дані асинхронно, що означає, що немає синхросигналу для синхронізації вихідних бітів від передавального UART до відбору бітів отримуючим UART. Замість тактового сигналу передавальний UART додає початок та зупинку бітів до переданого пакету даних. Ці біти визначають початок і кінець пакета даних, тому отримуючий UART знає, коли почати читання бітів даних.

Коли приймаючий UART виявляє початковий біт, він починає читати вхідний біти на певній частоті, відомі як швидкість передачі даних. Швидкість передачі даних - це міра швидкості передачі даних, виражена в бітах за секунду (bps). Обидва UART повинні працювати при приблизно однаковій швидкості передачі. Швидкість передачі та приймання UART може відрізнятися лише приблизно на 10%. Обидва UART також повинні бути налаштовані для передачі та отримання однакової структури пакетів даних.

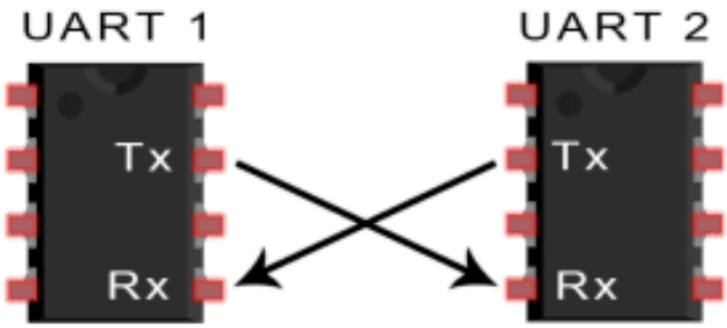

Рисунок 2.3 – Передача даних через UART

UART, який збирається передавати дані, отримує дані з шини даних. Шина даних використовується для передачі даних в UART іншим пристроєм, як процесор, пам'ять або мікроконтроллер. Дані передаються з шини даних на передачу UART у паралельній формі. Після передачі UART отримує паралельні дані з шини даних, він додає початковий біт, біт парності та стоп-біт, створюючи пакет даних. Далі, пакет даних виводиться послідовно, по біту на Tx пін. Приймаючий UART зчитує біт пакету даних один за одним з піну Rx. Приймаючий UART потім перетворює дані назад у паралельну форму та видаляє стартовий біт, біт парності та зупинки. Нарешті, отримуючий UART передає пакет даних паралельно шині даних на приймальному кінці.

Передані дані UART організовані в пакети. Кожен пакет містить 1 початкового біта, 5 - 9 бітів даних (залежно від UART), необов'язковий біт паритету та 1 або 2 біти зупинки (рис 2.4).

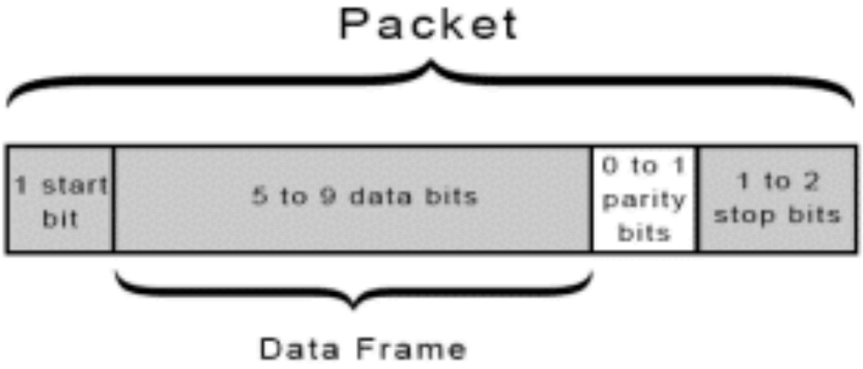

Рисунок 2.4 – структура пакету UART

Стартовий біт необхідний, оскільки лінія передачі даних UART зазвичай зберігається на рівні високої напруги, коли вона не передає дані. Щоб розпочати передачу даних, передавальний UART витягує лінію електропередачі від високої до низької протягом одного тактового циклу. Коли приймаючий UART виявляє перехід від високої до низької напруги, він починає читати біти в кадрі даних на частоті швидкості передачі.

Фрейм даних містить фактичні дані, що передаються. Це може бути 5 біт до 8 біт, якщо використовується біт парності. Якщо не використовується біт парності, кадр даних може бути дев'ятибітовий. У більшості випадків дані спочатку надсилаються найменшим значущим бітом.

Фрейм парності описує рівність або непарність числа. Біт парності - це спосіб отримання UART, щоб визначити, чи змінилися будь-які дані під час передачі. Біти можна змінювати за допомогою електромагнітного випромінювання, невідповідності швидкості передачі даних або передачі даних на великі відстані. Після того як приймач UART зчитує кадр даних, він розраховує кількість бітів з значенням 1 і перевіряє, чи загальний - рівний або непарний номер. Якщо біт парності є 0 (рівний паритет), 1 біт у кадрі даних повинен дорівнювати рівному числу. Якщо біт парності має значення 1 (непарне співвідношення), 1 біт у кадрі даних має складатися з непарним числом. Коли біт паритету відповідає даним, UART знає, що в передачі не було помилок.

Для сигналу про закінчення пакета даних, відправляючий UART приводить лінію передачі даних від низької напруги до високої напруги принаймні на два такти передачі бітів.

Алгоритм передачі даних через шину даних UART:

1. Передавальний UART одержує дані паралельно з шини даних;
2. Передаючий UART додає початковий біт, біт парності та стоп-біт до кадру даних;

3. Весь пакет подається серійно від передавального UART до приймаючого UART. Приймаючий UART знімає лінію даних за попередньо налаштованою швидкістю передачі;
4. Приймаючий UART відкидає початковий біт, біт парності та стоп-біт з кадру даних;
5. Приймаюча UART перетворює серійні дані назад паралельно і передає їх на шину даних на приймальному кінці.

Ні один протокол зв'язку не є ідеальним, але UART вважається одним з найнадійніших і розповсюджуваніших протоколів прийому та передачі даних. Перевагами використання даного протоколу є:

- Для запезпечення передачі достатньо лише двох дротів (Rx та Tx);
- Немає подтреби використання тактового сигналу;
- Має біт парності, щоб дозволити перевірку помилок;
- Структура пакета даних може бути змінена;
- Добре задокументований та широко використовуваний метод.

Недоліками використання даного протоколу є:

- Розмір кадру даних обмежений максимум дев'ятьма бітами;
- Не підтримує систему ведучий-ведений;
- Швидкість передачі кожного UART повинна бути не більше 10% один від одного;

SPI

Протокол послідовного периферійного інтерфейсу (SPI) - це асинхронний стандарт послідовних даних, який в основному використовується для того, щоб мікроконтролер мав можливість спілкуватися з іншими мікроконтролерами або іншими пристроями, такими як енергонезалежна пам'ять, запам'ятовуючі рідкокристалічні діоди (LCD), підсистеми аналого-цифрового перетворення тощо [14].

SPI - це простий протокол типу ведучий-ведений, що базується на чотирьох елементах:

- Clock line (SCLK)
- Serial output (MOSI)
- Serial input (MISO)
- Slave select (SS)

Кожна система SPI складається з одного ведучого та одного або декількох ведених, де ведучий ініціює зв'язок, затверджуючи лінію SS (вибір веденого). Коли вибрано підлеглий пристрій, ведучий починає передавати дані через лінію MOSI до вибраного підлеглого пристрою. Ведучий відправляє та отримує один біт для кожного такту таймера.

SPI - це примітивний протокол без механізму підтвердження для перевірки отриманих або відправлених даних. Для безпечного спілкування, керування потоком має бути впроваджено в комунікаційному протоколі на більш високому рівні.

Поширений послідовний порт, який належить до ліній TX та RX, називається "асинхронним", оскільки відсутній контроль над тим, коли надсилаються дані, або будь-яка гарантія того, що обидві сторони працюють точно з тією ж швидкістю. Оскільки комп'ютери зазвичай покладаються на все, що синхронізується з одним "годинником", це може бути проблемою, коли дві системи з дещо іншими "годинниками" намагаються зв'язатися один з одним.

Щоб вирішити цю проблему, асинхронні послідовні з'єднання використовують додаткові початкові та стоп-біти для кожного байта, вони допомагають синхронізувати приймач з даними, коли вони з'являються. Обидві сторони також повинні узгодити швидкість передачі (наприклад, 9600 біт на секунду) заздалегідь. Незначні відмінності в швидкості передачі не є проблемою, оскільки приймач повторно синхронізується на початку кожного байта.

Асинхронні серійні пристрої чудово працюють, але мають великі накладні витрати як на додаткові стартові зупинки, так і на зупинки, які надсилаються з кожним байтом, і на складне обладнання, необхідне для надсилання та отримання даних. І якщо обидві сторони не встановлені на тій же швидкості, отримані дані будуть некоректними. Це відбувається тому, що приймач відбирає біти в дуже конкретні часи.

SPI працює інакше. Це "синхронна" шина даних, що означає, що він використовує окремі лінії для даних і "годинник", який підтримує обидві сторони в синхронізації (рис. 2.5). Годинник - це сигнал, який точно повідомляє одержувачу вибірку бітів на лінії передачі даних. Це може бути перехід від низького до високого або перехід від високого до низького рівнів тактового сигналу. Коли приймач виявляє цей рівень, він негайно звертається на лінію даних, щоб прочитати наступний. Оскільки годинники надсилаються разом із даними, визначення швидкості не є важливим, хоча пристрої матимуть максимальну швидкість, за якою вони зможуть працювати.

Одна з причин того, що SPI широковикористовуваний це те, що приймаюче обладнання може бути простою системою зсуву.

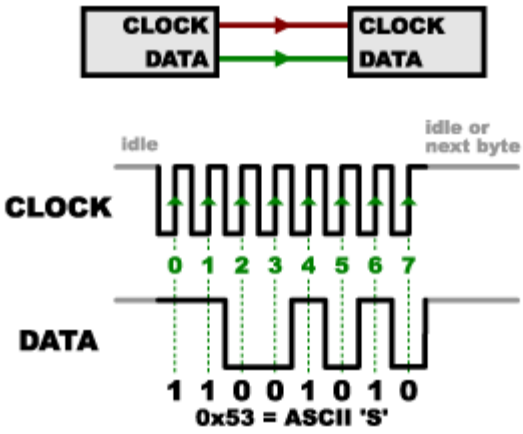

Рисунок 2.5 – Принцип роботи шини даних SPI

У SPI тільки одна сторона генерує тактовий сигнал (зазвичай називається CLK або SCK для Serial ClocK). Сторона, яка генерує годинник, називається "master" (ведучий), а інша сторона називається "slave"

(ведений). В SPI завжди є лише один ведучий (який майже завжди є мікроконтролером), але може бути декілька ведених.

Коли дані передаються від ведучого до веденого, він надсилається на лінії передавання даних, що називається MOSI (Master Out / Slave In). Якщо ведений повинен відправити відповідь назад, ведучий буде продовжувати генерувати попередньо встановлену кількість тактових циклів, а ведений поставить дані на третю лінію даних, що називається MISO (Master In / Slave Out).

Оскільки ведучий завжди генерує тактовий сигнал, він повинен знати заздалегідь, коли ведений повинен повернути дані та скільки даних буде повернуто. Це дуже відрізняється від асинхронного порту, де випадкові величини даних можуть бути відправлені в будь-якому напрямку в будь-який час. SPI зазвичай використовується для комунікації з датчиками, що мають дуже специфічну командну структуру.

SPI – є повнодуплексним (має окремі лінії відправлення та отримання), і, таким чином, в певних ситуаціях можливо передавати та отримувати дані одночасно (наприклад, запитуючи новий датчик, читаючи отримані дані з попереднього).

Лінія SS, як правило, утримується у високому рівні, що відключає веденого від шини SPI. Перед тим, як дані надсилаються до веденого, лінія переходить на низький рівень, що активує його. Коли закінчиться його використання, лінія знову поверняться у високий стан.

Кожному ведучому потрібна окрема лінія SS (рис 2.6). Щоб налагодити зв'язок з конкретним веденим, відбувається перехід SS на низький рівень, а решта залишаються у високому рівні.

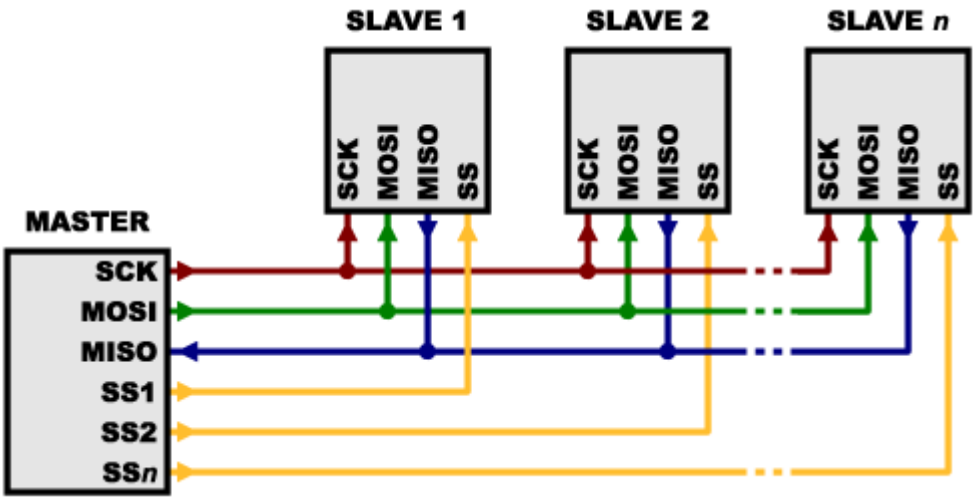

Рисунок 2.6 – SPI з декількома веденими

Багато мікроконтролерів мають вбудовані периферійні пристрої SPI, які обробляють всі деталі надсилання та отримання даних, і можуть робити це на дуже високих швидкостях.

Під час налаштування інтерфейсу потрібно вибрати деякі параметри. Ці параметри повинні відповідати параметрам мікроконтролеру, на якому є SPI:

- Інтерфейс може передавати дані першим або найменш значущим бітом (LSB) або найважливішим бітом (MSB).
- Ведений буде читати дані на передньому або задньому фронтах тактового імпульсу. Крім того, "годинник" може вважатися неактивним, коли рівень імпульсу високий або низький.
- SPI може працювати на дуже високих швидкостях (мільйони байтів у секунду), що може бути надто швидким для деяких пристроїв. Щоб розмістити такі пристрої, потрібно налаштувати швидкість передачі даних.
- Потрібно налаштувати піни SCK, MOSI та MISO.
- Піном SS необхідно керувати самостійно, а саме встановлювати низький рівень перед передачею даних та встановлювати високий після неї.

Перевагами шини даних SPI є:

- Цей протокол швидший за асинхронний;
- Пристроєм для прийому даних може бути простий регістр зсуву;
- Даний протокол може мати велику кількох ведених.

Недоліками даного протоколу є:

- Він вимагає більше сигнальних ліній (проводів), ніж інші способи зв'язку;
- Комунікації повинні бути чітко визначені заздалегідь (не можна відправляти випадкові обсяги даних, коли завгодно);
- Ведучий повинен контролювати всі комунікації;
- Зазвичай для кожного веденого вимагається окрема лінія SS, що може бути проблематичним, якщо їхня кількість велика.

CAN

Протокол CAN був розроблений компанією BOSCH як багатопрофільна система передачі повідомлень, яка визначає максимальну швидкість передачі сигналу – один мегабіт в секунду (біт / с) [15]. На відміну від традиційної мережі, такої як USB або Ethernet, CAN не надсилає великі блоки даних від вузла А до вузла В під наглядом центральної шини. У мережі CAN багато коротких повідомлень, таких як температура, і вони передаються по всій мережі, що забезпечує узгодженість даних у кожному вузлі системи.

CAN – це послідовна комунікаційна шина, визначена Міжнародною організацією стандартизації (ISO), спочатку була розроблена для автомобільної промисловості для заміни складної проводки з двопроводною шиною. Специфікація вимагає високої імунітету до електричних перешкод і здатності самодіагностики та відновлення помилок даних. Ці особливості призвели до популярності CAN в різних галузях промисловості, включаючи автоматизацію будівництва, медичну та виробничу діяльності.

Протокол CAN-зв'язку - це протокол багаторазового доступу носія, який має виявлення зіткнень і арбітраж на пріоритеті повідомлень (CSMA /

CD + AMP). CSMA означає, що кожен вузол на шині повинен чекати встановленого періоду бездіяльності перед спробою відправити повідомлення. CD + AMP означає, що зіткнення вирішуються через бітовий арбітраж, заснований на попередньо запрограмованому пріоритеті кожного повідомлення в полі ідентифікатора повідомлення. Ідентифікатор вищого пріоритету завжди отримує доступ до шини. Оскільки кожен вузол на шині бере участь у передачі кожного біту, арбітражний вузол знає, коли на шині розміщений біт логічної одиниці.

Є два види повідомлень, що передаються по шині CAN. Перший тип повідомлення є стандартним одинадцятибітним повідомленням (рис 2.7).

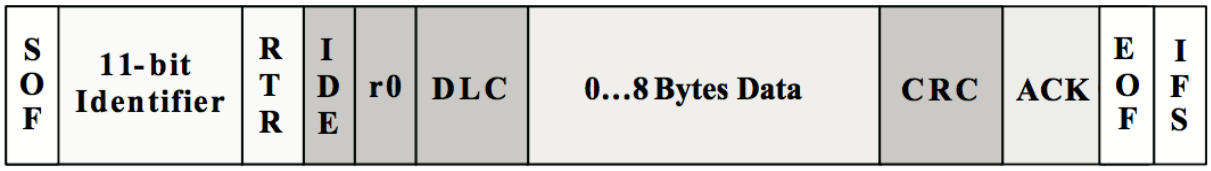

Рисунок 2.7 – Стандартне повідомлення CAN

Значеннями полів цього повідомлення є:

- SOF (start of frame). Біт єдиного домінантного початку повідомлення позначає початок повідомлення, і він використовується для синхронізації вузлів на шині після простою.
- Ідентифікатор. Стандартний 11-бітний ідентифікатор CAN встановлює пріоритет повідомлення. Чим нижче двійкове значення, тим вище його пріоритет.
- RTR (remote transmission request). Один біт передачі запиту на віддалену передачу є домінуючим, коли інформація потрібна з іншого вузла. Всі вузли отримують запит, але ідентифікатор визначає вказаний вузол. Відповідні дані також отримуються всіма вузлами і використовуються будь-яким вузлом, який зацікавлений в них. Таким чином, всі дані, що використовуються в системі, є однорідними.
- IDE. Розширення ідентифікатора IDE-A означає, що передаються стандартний ідентифікатор CAN без розширення.

- r0. Зарезервований біт (для можливого використання майбутньою стандартною зміною).
- DLC (data length code). 4-бітний код довжини даних містить кількість байт переданих даних.
- Місце повідомлення відведене для передачі до 64 біт даних.
- CRC. Перевірка циклічного надлишкового коду з 16 біт (15 біт плюс роздільник) містить контрольну суму (кількість бітів, що передаються) попередніх даних програми для виявлення помилок.
- ACK (acknowledges). Кожен вузол, який отримує точне повідомлення, перезаписує цей рецесивний біт у початковому повідомленні з доменним бітом, який вказує на повідомлення без повідомлення про помилку. Якщо приймальний вузол виявляє помилку і залишає цей біт рецессивним, він відкидає повідомлення, а відправний вузол повторить повідомлення після повторної активації. Таким чином, кожен вузол визнає ACK цілісність своїх даних.
- EOF (end of frame). Кінцевий кадр, що є 7-бітовим полем і позначає кінець CAN-кадру (повідомлення) і вимикає бітове накладення. Коли 5 бітів одного й того ж логічного рівня послідовно відбуваються під час звичайної роботи, в дані вставляється протилежний рівень логіки.
- IFS (interface space). 7-бітни1 інтерфейс, що містить час, необхідний мыкроконтролера для переміщення правильно прийнятого кадру до його належної позиції в області буферу повідомлень.

Другим типом повідомлення в шині даних CAN є розширене повідомлення (рис 2.8) і воно відрізняється тим що містить додаткові відділи, такі як:

- SRR (substitute remote request). Біт заміщення віддаленого запиту замінює біт RTR у стандартному повідомленні як заповнювач у розширеному форматі;

- Рецесивний біт IDE-A у розширенні ідентифікатора IDE вказує, що наступні біти ідентифікатора відповідають розширеному типу повідомлення;
- r1. Після бітів RTR і r0 додатковий запасний біт був включений до біта DLC.

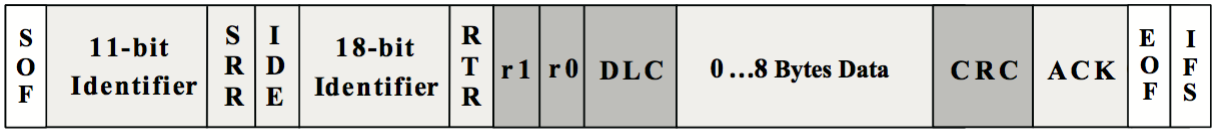

Рисунок 2.8 - Розширене повідомлення CAN

Фундаментальна характеристика CAN (рис 2.9) - це протилежний логічний стан між шиною та вихідний сигнал драйвера та приймача. Як правило, висока логіка асоціюється з логічною одиницею, а низька логіка асоціюється з логічним нулем, але на шині CAN це не так. Саме тому прийомні передатчики CAN мають вхідний сигнал драйвера, так що при відсутності будь-якого входу пристрій автоматично за замовчуванням встановлює стан рецесійної шини на всіх входах та виходах.

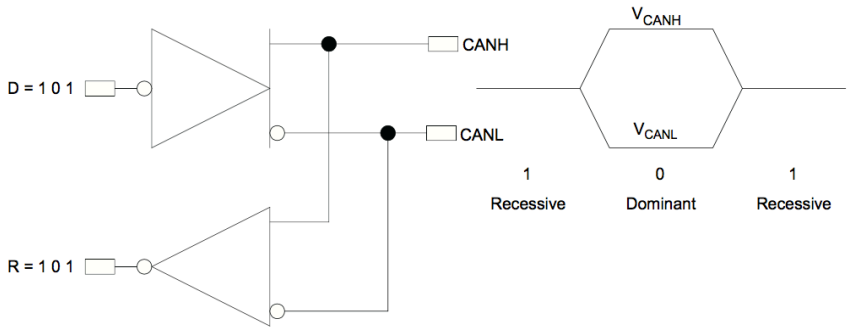

Рисунок 2.9 – Інверсна логіка CAN шини

Доступ до шини залежить від подій і відбувається випадковим чином. Якщо два вузли намагаються одночасно зайняти шину, доступ здійснюється за допомогою неруйнівного, бітового арбітражу.

Визначення пріоритету повідомленням в ідентифікаторі є функцією CAN, що робить його особливо зручним для використання в режимі реального часу. Чим нижче двійковий номер ідентифікатора повідомлення, тим вище його пріоритет. Ідентифікатор, що складається цілком з нулів, є

найвищим пріоритетом у мережі, оскільки він тримає домінуючу шину найдовше. Тому, якщо два вузли починають передавати одночасно, вузол, який надсилає останній біт ідентифікатора як логічний нуль (домінуючий), тоді як інші вузли відправляють логічну одиницю (рецесивний), зберігають контроль над шиною CAN і продовжують заповнювати своє повідомлення. Домінуючий біт завжди перезаписує рецесивний біт на шині CAN.

Передавальний вузол постійно контролює кожен біт своєї власної передачі. Це причина для конфігурації прийомного приймач, в якій CANH та CANL внутрішньо прив'язані до входу приймача. Затримка поширення сигналу у внутрішньому циклі від вхідного драйвера до виходу приймача зазвичай використовується як якісна міра трансивера CAN. Ця затримка розповсюдження називається циклом часу, але набуває різноманітну номенклатуру від різних постачальників.

На рис. 2.10 показаний арбітражний процес CAN, який автоматично обробляється контролером CAN. Оскільки кожен вузол постійно контролює власні передачі, рецесивний біт вузла B перезаписується доменним бітом старшого пріоритету вузла C, і виявляє, що стан шини не відповідає біту, який він передавав. Тому вузол B зупиняє передачу, а вузол C продовжує своє повідомлення. Ще одна спроба передачі повідомлення здійснюється вузлом B, коли шина відпускається вузлом C.

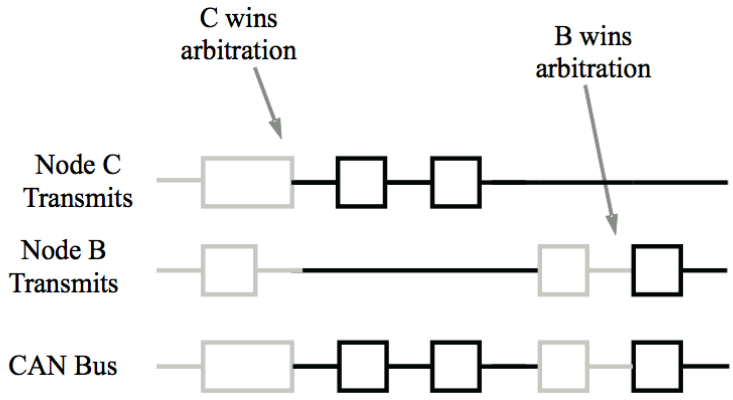

Рисунок 2.10 – Арбітраж на шині CAN

Є чотири різних типи повідомлень або фреймів, які можуть передаватися на шині CAN:

- Повідомлення даних. Повідомлення даних є найпоширенішим типом повідомлення, і включає поле арбітражу, поле даних, поле CRC і поле підтвердження. Арбітражне поле містить 11-бітний ідентифікатор та біт RTR, який домінує для кадрів даних. Він містить 29-бітний ідентифікатор та біт RTR. Наступним є поле даних, яке містить від 0 до 8 байт даних, і поле CRC, яке містить 16-бітну контрольну суму, яка використовується для виявлення помилок.
- Віддалене повідомлення. Призначенням віддаленого фрейму є запит на передачу даних з іншого вузла. Віддалене повідомлення схоже на повідомлення даних з двома важливими відмінностями. По-перше, цей тип повідомлення явним чином позначається як віддалений фрейм рецессивним бітом RTR у полі арбітражу, по-друге, немає даних.
- Повідомлення помилки. Фрейм помилки - це спеціальне повідомлення, яке порушує правила форматування повідомлення CAN. Воно передається, коли вузол виявляє помилку в повідомленні та закликає всі інші вузли в мережі також відправити повідомлення помилок. Передавач автоматично повторює повідомлення. Розроблена система лічильників помилок в контролері CAN забезпечує те, що вузол не може зв'язати шину, багаторазово передаючи кадри помилок.
- Повідомлення перевантаження. Фрейм перевантаження згадується для повноти. Воно схоже на повідомлення помилки щодо формату і передається вузлом, який надто зайнятий. Це, перш за все використовується для забезпечення затримки між повідомленнями.

2.3 Енергонезалежна пам'ять

Кожний апаратний пристрій повинен мати змогу динамічно переналаштовуватись працівником. Також після увімкнення йому необхідно продовжувати роботу відповідно до попередніх налаштувань. Для реалізації цих функцій виконавчої системи засобів автоматизації складських виробничих процесів було вирішено використовувати енергонезалежну пам'ять EEPROM M25P40, яка є зручною завдяки можливості посторінкового запису, а також системи команд для роботи з нею.

M25P40 - це серійна флеш-пам'ять розміром 4 мегабайти з сучасними механізмами захисту від запису, доступ до яких забезпечується високошвидкісною шиною, сумісною з SPI [16]. Пристрій підтримує високопродуктивні команди для тактової частоти до 75 МГц.

Пам’ять має 8 входів (рис. 2.11):

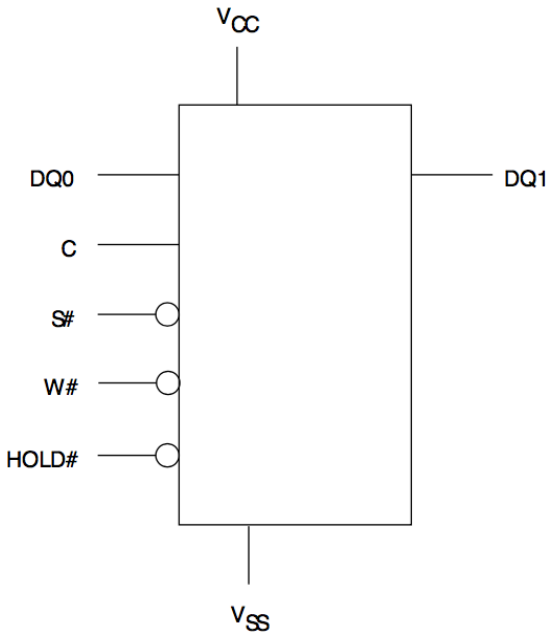

Рисунок 2.11 – Логічна схема M25P40

- DQ1. Вихідний сигнал DQ1 використовується для передачі даних послідовним способом. Дані зміщуються по задньому фронту системного годинника C.
- DQ0. Вхідний сигнал DQ0 використовується для передачі даних послідовним способом у пристрій. Він отримує команди, адреси та

дані, які потрібно запрограмувати. Значення фіксуються на передньому фронті краю системного годинника C.

- C. Вхідний сигнал C забезпечує синхронізацію послідовного інтерфейсу. Команди, атрибути або дані, присутні при введенні послідовних даних DQ0, фіксуються на передньому фронті системного годинника C. Дані щодо DQ1 змінюються після по задньому фронту системного годинника.
- S#. Коли цей сигнал високий, пристрій пам’яті не буде обрано, а рівень DQ1 буде високим. Якщо внутрішні команди PROGRAM, ERASE або WRITE STATUS не працюють, пристрій перебуватиме в режимі очікування. Подача на S# низького рівня дозволяє пристрою перейти в активний режим живлення. Після активації до початку відправлення будь-якої команди S# має перейти по задньому фронту системного годинника.
- HOLD#. Цей Сигнал використовується для призупинення будь-яких послідовних комунікацій з пристроєм.
- Vcc та Vss. Сигнал живлення та землі відповідно.
- Vpp. Це як керований вхід, так і канал живлення. Дві функції вибираються діапазоном напруги, що застосовується до каналу. Якщо вхід Vpp зберігається в діапазоні низької напруги (від 0 V до VCC), канал розглядається як контрольний ввід. Вхідний сигнал Vpp використовується для зупигки зростання розміру пам'яті, захищеної від програм або стирання команд. VPP виступає в ролі додаткового джерела живлення, якщо він знаходиться в діапазоні високого рівня.

Пам'ять може бути запрограмована від 1 до 256 байтів одночасно за допомогою команди PAGE PROGRAM. Вона організована як 8 секторів, кожен з яких містить 256 сторінок. Кожна сторінка дозволяє зарис до 256 байт.

Вся пам'ять може бути стерта за допомогою команди BULK ERASE або одночасно можна стерти один сектор за допомогою команди SECTOR ERASE.

2.3 Програмні засоби розробки

Мікроконтролер на основі ядра ARM Cortex M0 дозволяє виконувати розробку програмного забезпечення на високорівневих мовах програмування. В якості такої мови програмування обрана мова С, оскільки вона забезпечує швидке перетворення та формування оптимальноого вихідного коду. Також були додані вставки мовою асемблера для оптимізації роботи системи.

Виконавчий код програмного забезпеченння скомпільовано у віртуальному середовищі IAR Embedded Workbench, оскільки воно містить оптимізований для роботи вбудованих систем компілятор.

Мова програмування С

Мова програмування С є високорівневою мовою, що була створена для розробки програмного забезпечення, хоча спочатку вона була призначена для написання виключно системного програмного забезпечення [17].

С належить до структурованих, процедурних парадигм мов. Це гнучка і потужна мова програмування і вона може використовуватися для різних застосувань. Незважаючи на те що С є високорівневою мовою, вона має багато однакових атрибутів з мовою асемблеру, що збільшує її ефективність і швидкодію.

Одним з підвидів мови програмування С є Embedded C. Вона складається з мовних наборів С, які можуть бути використані для різних цілей. Вона в основному використовує стандартні синтаксис і семантику. Embedded C є широко використовуваною мовою програмування, оскільки

більшість вбудованих систем, таких як наприклад мобільні телефони, сучасні автомобілі, мікроконтролери тощо розроблюються на ній.

IAR Embedded Workbench

IAR Embedded Workbench - це потужне інтегроване середовище розробки (IDE), яке дозволяє розробляти та управляти повними вбудованими проектами додатків. Це забезпечує просте в освоєнні та високоефективне середовище розробки з максимальними можливостями успадкування коду, комплексною та спеціальною цільовою підтримкою. IAR Embedded Workbench пропагує корисну робочу методологію і тим самим суттєво скорочує час розробки.

Сердовище розробки IAR містить власний компілятор для ARM, який пропонує стандартні функції мов C та C ++, а також розширення, призначені для використання переваг ARM-специфічних об'єктів. Код розроблений на цих мовах після компіляції конвертується в мову асемблера. Компілятор IAR являється одним з найпопулярнішим компіляторів завдяки своїй оптимізації (таб. 2.1) [18]. Як видно з таблиці 2.1 IAR створює менший за розміром та швидший образ програми, в порівнянні з компілятором GCC.

Таблиця 2.1 – Порівняння розміру і швидкодії образу програми скомпільованої компіляторами GCC, NTXGCC та IAR

Компілятор	Розмір коду	Розмір RAM	Швидкість при використанні benchmark
GCC	202052	54548	615
NTXGCC	180104	54144	-
IAR	122440	49831	640

Мова асемблера IAR для ARM - це потужний макро асемблер з універсальним набором директив та операторів. Мова асемблера має вбудований препроцесор мови C і підтримує умовну збірку. Вона

використовує той же синтаксис мнемоніки та операндів, що і Advanced RISC Machines Assembler, що спрощує міграцію існуючого коду.

Ще одним важливим етапом створення програмного забезпечення в IAR є компонувальник. Компонувальником є програма, що приймає на вхід один або кілька об'єктних модулів і збирає їх в один виконуваний модуль. Компонувальник IAR є гнучким програмним інструментом для розробки мікроконтролерів. Він також чудово підходить як для компонування невеликих однофайлових абсолютних асемблерних програм, так і для компонування великих модулів на мовах C або C++ або змішаних програм написаних на C або C++ та мові асемблеру.

Висновки до розділу 2

Здійснено поділ розроблених засобів автоматизації складських виробничих процесів на дві складові: виконавчу та керуючу системи. Запропоновано реалізувати виконавчу систему сукупністю програмно-апаратних пристроїв на основі мікроконтролерів Cortex ARM M0, вибір яких обгрунтовано.

Основною шиною даних для комунікації з системою обрано інтерфейс CAN, який відрізняється простотою налаштування та використання шини, а також можливістю використовувати розширені пакети даних для передачі.

В якості шини даних для налаштування мікроконтролера обрано шину даних UART. Цей інтерфейс простий в реалізації його налаштування, а також існує достатньо програм для комунікації з даною шиною даних.

Для можливості динамічного налаштування мікроконтролера та збереження його налаштувань запропоновано використати енергонезалежну пам’ять EEPROM M25P40 та шину даних SPI для роботи з нею.

3. РЕАЛІЗАЦІЯ ВИКОНАВЧОЇ СИСТЕМИ ЗАСОБІВ АВТОМАТИЗАЦІЇ СКЛАДСЬКИХ ВИРОБНИЧИХ ПРОЦЕСІВ

3.1. Основні компоненти системи автоматизації складських виробничих процесів

Розроблювані засоби моніторингу складських виробничих процесів поділяється на дві взаємозв'язані частини (рис. 3.1): виконавчу систему, яка є системою пристроїв (мікроконтролерів) та керуючу системи.

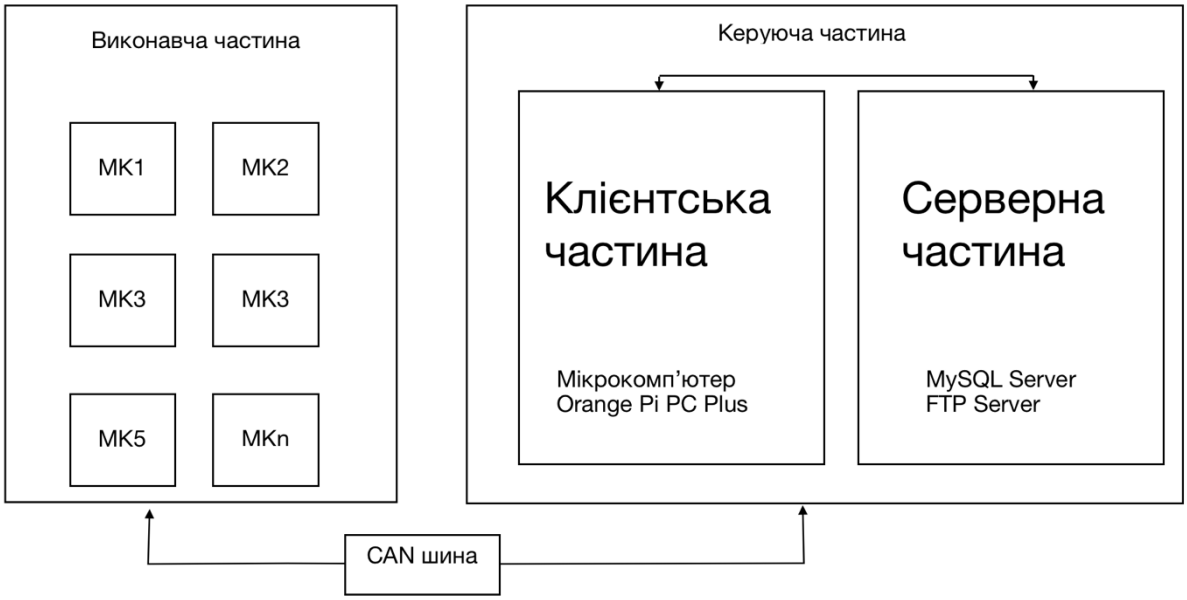

Рисунок 3.1 – Керуюча та виконавча системи

Керуюча частина засобів автоматизації визначає задачу, яку потрібно вирішити, та алгоритм роботи виконавчої системи і виконує такі функції:

1. Налаштування системи і вибір задачі для виконання;
2. Узгодження та відправка команд виконавчій частині;
3. Візуальне відображення роботи системи;
4. Додавання нових задач в систему;
5. Надання корисних інструкцій, як має бути виконаний кожний етап задачі;

6. Налаштування пристроїв виконавчої системи на необхідний режим роботи;
7. Моніторинг роботи працівника.

Для виконавчої системи передбачено виконання наступних функцій:

1. Ініціалізація системи, завантаження поточних налаштувань;
2. Ініціалізація елементів пристрою, які призначені для роботи з працівниками.
3. Ініціалізація аналізатора вхідних команд від керуючої системи для забезпечення зв'язку з виконавчою системою;
4. Отримання та відправлення даних в порти введення/виведення;
5. Отримання та відправлення даних через визначену шину даних.

Процес обміну інформації між керуючою та виконавчою системами відбувається через інтерфейс CAN. Кожний мікроконтролер має свій ідентифікатор, який є також його адресою на шині даних. Оскільки в цьому інтерфейсі реалізовано принцип арбітражу, кожний пристрій має мати свій унікальний ідентифікатор, щоб унеможливити виникнення помилки на шині при одинакових адресах мікроконтролера і некоректної роботи системи.

Кожний пристрій протягом визначеного інтервалу часу відправляє дані певного формату (паспорт пристрою, з яким здійснюється роботи в поточний час):

```
void CAN_Transmit_PASPORT (void) {
  register Type_Global_Counter*  p_Global_Counter =
&Global_Counter;
  uint32_t Pasport_Data[2];
  Pasport_Data[0] = *(HW_Ver);
  Pasport_Data[1] = *(SW_Ver);
  CAN_Transmit(Pasport_Data, CAN_PASSPORT_SEND);
  CAN_Transmit(p_Global_Counter->Counter,CAN_TRANSMIT);
  p_Global_Counter->Run_time+=5;
}
```

Це відбувається для того, щоб керуюча система перевірила можливість підтримування зв'язку з кожним пристроєм, який необхідний

для виконання поточної роботи і, у випадку збою, могла повідомити працівнику про стан кожного з пристроїв.

Для ініціалізії інтерфейсу CAN, окрім внесення відповідних бітів в регістри налаштування роботи, також необхідно реалізувати налаштування прийому та відправки повідомлень через цю шину даних. Оскільки при першому ввімкненні пристрою він завантажує налаштування за замовчуванням, реалізується динамічна зміна його адреси та тих повідомлень, які він буде відправляти за новою адресою. Для цього реалізується наступна функція:

```
        void Message_Init(uint32_t Adress, uint32_t
Number_unic_filter_mesage,...)
        {
         register const Type_Message_Option* Temp_Message  =
Message_Option;
         register       uint32_t*     Message_count = &Number_unic_filter_mesage;
          register       uint32_t*           Register_Can  = CAN_BASE;
          for(uint32_t i=MESSAGE_QUANTITY; i--; Temp_Message++)
              {
                if(Temp_Message->Message_Num == *Message_count)
                    {
                       Adress = *(Message_count+1);
                       Message_count+=2;
                    }
        Register_Can[CAN_F1_CMDREQ+1]=(CTRL|ARB|MASK|WR);
        Register_Can[CAN_F1_CMDREQ+2]=Temp_Message->Filter_Mask_1;
        Register_Can[CAN_F1_CMDREQ+3]=Temp_Message->Filter_Mask_2;
        Register_Can[CAN_F1_CMDREQ+4]=Temp_Message->ARB_1|Adress;
        Register_Can[CAN_F1_CMDREQ+5]=Temp_Message->ARB_2;
        Register_Can[CAN_F1_CMDREQ+6]=(DLC_MAX|EOB|UMSK|RXIE);
        Register_Can[CAN_F1_CMDREQ]  = Temp_Message->Message_Num;
        while (Register_Can[CAN_F1_CMDREQ] & IFCREQ_BUSY);
        }
    }
```

3.2 Операційна система

Створення простого пристрою, який прийматиме та оброблятиме дані, отримані з датчика, не складає великих зусиль. Натомість, створення складної системи пристроїв, що оброблюватимуть різні потоки даних з

різних носіїв як аналогової так і цифрової інформації, та на різних кроках обробки прийматимуть різні рішення ускладнює задачу. Тому для розробки виконавчої частини засобів для автоматизації складських виробничих процесів використовувалась операційна система реального часу (ОСРЧ).

ОСРЧ називається така система, що обслуговує додатки реального часу, які обробляють дані по мірі їх надходження, як правило, без затримок в черзі надходжень [19].

Основними причинами використання ОСРЧ є структуризація написаного коду, а також, що є основним, можливість розпаралелювання задач, виконуваних виконавчою системою.

Кожній задачі присвоюється унікальний рівень пріоритету (від 0 до n, де n є останнім номером задачі). Задача n є задачею з найнижчим пріоритетом. Задачею з найнижчим пріоритетом є завжди IDLE задача, що досліджує чергу інших процесів і при відповідних умовах зупиняє або запускає їх:

```
for(uint32_t i=0; i<TABLE_TASK_QUANT; i++, p_Task_Table++)
      {
        flag_task_on =0;
        pointer = (uint32_t*)&p_Task_Table->timer;
         if(timestramp && *pointer)
             {
                 if(*pointer>timestramp) *pointer-=timestramp;
                 else
                    {
                       *pointer = 0;
                        flag_task_on = 1;
                    }
             }
         if(p_Task_Table->flag ||  flag_task_on)  TaskOn(p_Task_Table->task_num);
        }
```

Для роботи з процесами використовуються таблиця пріоритетності RdyTbl[]. Для визначення пріоритету, а отже, і задачі, що виконуватиметься далі, планувальник ОСРЧ визначає найменший пріоритетний номер, що має біт, встановлений в RdyTbl []:

const unsigned char OSMapTbl[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

OSRdyGrp |= OSMapTbl[prio >> 3];

OSRdyTbl[prio >> 3] |= OSMapTbl[prio & 0x07];

Зв'язок між RdyGrp та RdyTbl [] показаний на рис. 3.3.

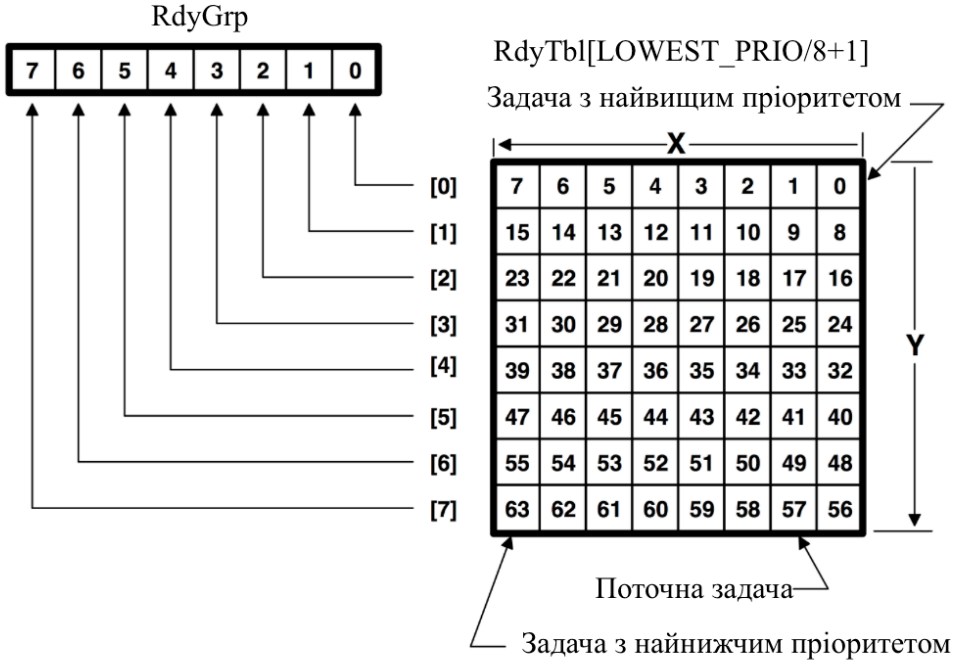

Рисунок 3.3 – Визначення пріоритету задач

Задача видаляється з списку готових задач інверсним способом:

```
if ((OSRdyTbl[prio >> 3] &amp;= ~OSMapTbl[prio & 0x07]) == 0)
    OSRdyGrp &= ~OSMapTbl[prio >> 3];
```

Цей код очищає готовий біт завдання в RdyTbl [] і очищує біт у RdyGrp, лише якщо всі завдання в групі не готові до запуску, тобто, всі біти в RdyTbl [prio >> 3] дорівнюють 0. Для пошуку готового до виконання завдання яке має найвищий пріоритет, виконується пошук за таблицею UnMapTbl. UnMapTbl [256] - це таблиця пріоритетних дозволів (рис 3.4). Вісім бітів відображаються, коли завдання в групі готові. Найменший біт має найвищий пріоритет. Використовуючи цей біт для індексації, UnMapTbl [] повертає бітову позицію набору найвищих пріоритетів - число між 0 та 7:

```
y=OSUnMapTbl[OSRdyGrp];
x=OSUnMapTbl[OSRdyTbl[y]];
prio = (y << 3)+x;
TaskRan = TaskList[prio];
```

```
const unsigned char UnMapTbl[] = {
                    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
                    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
                      };
```

Рисунок 3.4 – Масив пріоритетних дозволів

Наприклад, якщо RdyGrp містить 0x68, то пошук у таблицях UnMapTbl[RdyGrp] дає значення 3, яке відповідає бітові 3 в RdyGrp. Аналогічно, якщо RdyTbl[3] містить 0xE4, то UnMapTbl [RdyTbl [3]] дає значення 2 (біт 2). Пріоритет завдання (prio) становитиме тоді 26 (тобто 3 x 8 + 2).

Після визначення пріоритетів задач і заповнення таблиці RdyTbl, управління переходить до планувальника задач. В ОСРЧ виконавчої системи засобів автоматизації складських виробничих процесів роль планувальника виконує остання задача. Якщо при виконання задачі в чергу надійшла задача з вищим пріорітетом, тоді виконання задачі з нижчим пріоритетом буде призупинено. За коректне зупинення і відновлення задачі відповідає контекстний перемикач. Робота контекстного перемикача полягає в збереженні регістрів задачі, що підлягає призупиненню, в її стеку, і відновленні регістрів задачі з вищим пріоритетом зі свого стеку. Іншими словами, все, що повинен виконувати контекстний перемикач для виконання готової задачі - це відновити всі регістри процесора зі стеку задачі та виконати повернення з переривання (рис. 3.5). Весь код планувальника задач розглядається як критичний розділ. Тому в ньому забороняються всі переривання для запобігання встановлення біта

готовності для одного або декількох процесів під час пошуку найактуальнішої задачі, готової до виконання.

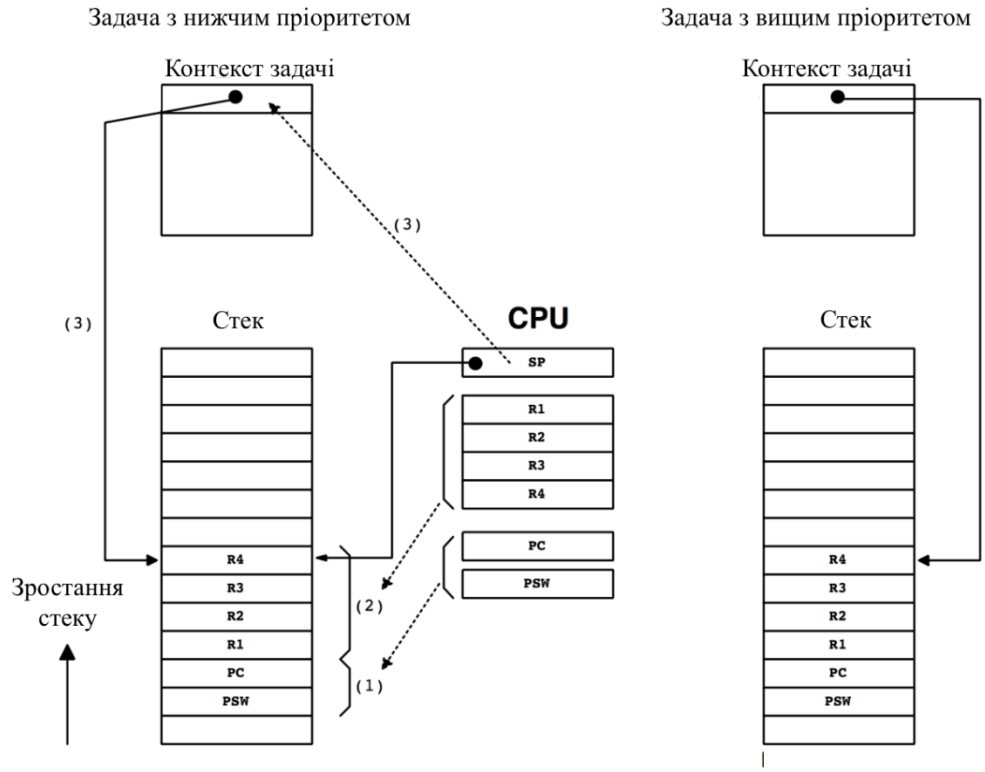

Рисунок 3.5 – Контекстний перемикач задач

Основною вимогою до ОСРЧ є тактування від єдиного таймеру. В розробленій операційній системи цю функцію виконує системний таймер (SysTick). SysTick є одним з найпростіших апаратних таймерів в мікроконтролерах версій Cortex Arm M0. Основною його задачею є генерування переривань у заданому проміжку часу.

На відміну від відомих ОСРЧ, ініціалізація задач розробленої операційної системи не потребує ніяких додаткових параметрів, таких як: назва задачі, динамічної зміни пріоритету задачі, ініціалізації callback функції задачі. Ця відмінність зумовлює зменшення використання оперативної пам’яті при роботі системи. Разом з додаванням ідентифікатора задачі в чергу планувальника, також додається її таймер (timer), стан (flag), та подія (event) (рис. 3.6).

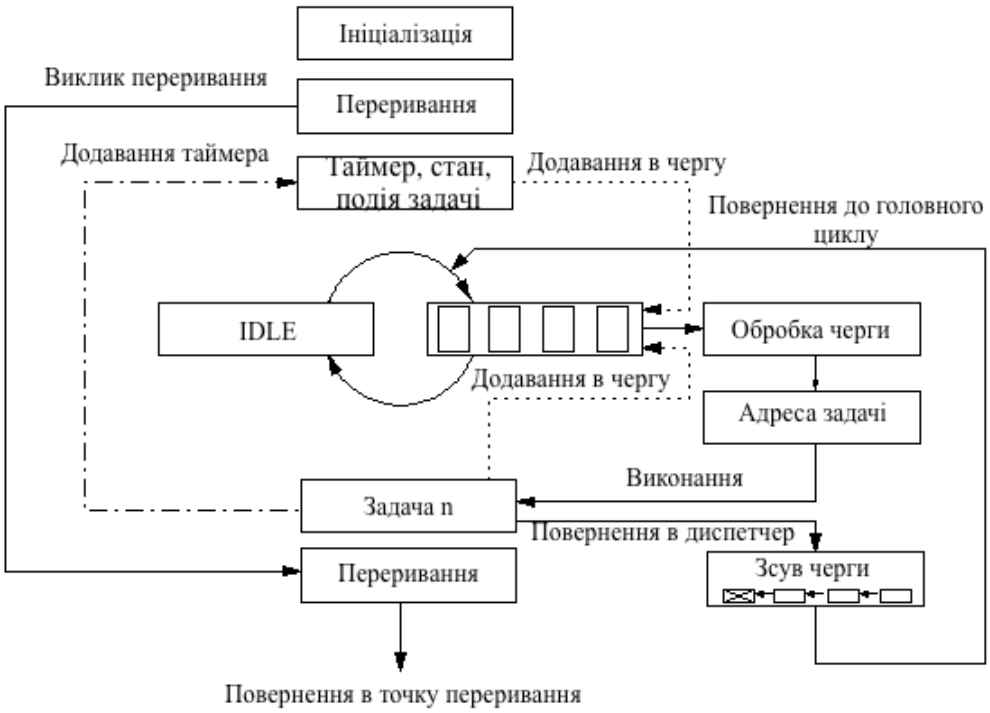

Рисунок 3.6 – Робота планувальника задач

Стан задачі перевіряється першочергово. Якщо він не дорівнює нулю, задача почне виконуватись одразу ж. При додаванні значення таймера, відмінного від нуля, задача почне виконуватись після того, як пройде час, зазначений у таймері. Подія задачі не може запустити задачу, але задача виконуватиме різні підзадачі при різних значеннях події.

ОСРЧ виконавчої системи, окрім IDLE задачі, виконує ще шість задач (процесів):

1. Процес ініціалізації системи, завантаження параметрів налаштувань з енергонезалежної пам’яті, налаштування повідомлень в інтерфейсі CAN;
2. Процес оброблення системних команд, надісланих користувачем, та відповіді на них;
3. Процес відправлення паспорта пристрою;
4. Процес функціонування пристрою в режимі паузи;
5. Процес функціонування пристрою в режимі створення нової збірки;
6. Процес функціонування пристрою в режимі виконання збірки.

3.3 Система параметрів апаратного пристрою виконавчої системи

Для забезпечення гнучкості системи автоматизації і можливості легкої заміни пошкоджених апаратних пристроїв, потрібно забезпечити динамічну реконфігурацію пристроїв. Це можливо завдяки створенню системи параметрів в кожному пристрої, які виконують функції при заданих значеннях.

Є чотири типи пареметрів якими може оперувати користувач:

1. Байтові параметри (від 0 до 255);
2. Двохбайтові параметри (від 0 до 65535);
3. Чотирьохбайтові параметри (від 0 до 4294967295);
4. Рядкові параметри (від 1 до 31 символу);

Перші три типи параметрів можуть бути поданими як в десятковій так і в шістнадцятковій системі числення. Ця можливість обирається при ініціалізації таблиці параметрів, представленої на рис. 3.7:

```
const SETTINGS_STRUCT REQUEST_SETTINGS[] = {
{ID_Reboot<<3     |_IN_DEC,   .MEMORY=WORD+BYTE, .PAGE=SYS_PAGE_1,   RAM.WORD[SYS_Reboot].WORD,     DEFAULT_VALUE.WORD[SYS_Reboot].WORD},
{ID_CAN_ADDR<<3   |_IN_DEC,   .MEMORY=BYTE+BYTE, .PAGE=PAGE_BYTE_1,  RAM.BYTE[CAN_ADDR].BYTE,       DEFAULT_VALUE.BYTE[CAN_ADDR].BYTE},
{ID_CAN_PERIOD<<3 |_IN_HEX,   .MEMORY=INT+BYTE,  .PAGE=PAGE_INT_1,   RAM.INT[CAN_PERIOD].INT,       DEFAULT_VALUE.INT[CAN_PERIOD].INT},
{ID_PASSWORD<<3   |_IN_STRING,.MEMORY=STRING,    .PAGE=PAGE_STRING_1,RAM.STRING[PASSWORD].STRING,   DEFAULT_VALUE.STRING[PASSWORD].STRING}
{ID_DEVICE_NAME<<3|_IN_STRING,.MEMORY=STRING,    .PAGE=PAGE_STRING_1,RAM.STRING[DEVICE_NAME].STRING,DEFAULT_VALUE.STRING[DEVICE_NAME].STRI
};
```

Рисунок 3.7 – Таблиця ініціалізації параметрів

Подана вище таблиця REQUEST_SETTINGS[] ініціалізує основні дані кожного параметру. Цими даними є:

1. Код ідентифікатора, зміщений на три біти вліво, де інформація про його подання (десяткове, шістнадцяткове, рядкове) знаходиться в перших трьох біти.
2. Пам'ять параметра. Для параметра в пам'яті виділяється один додатковий байт для циклічного надлишкового коду (CRC);
3. Сторінка. Кожен тип параметра зберігається в окремій сторінці енергонезалежної пам’яті;
4. Значення параметра;

5. Значення циклічного надлишкового коду параметра.

Циклічний надлишковий код визначає алгоритм знаходження контрольної суми, що необхідний для перевірки коректності даних. CRC є практичним додатком завадостійкого кодування, заснованого на математичних властивостях циклічного коду [20]. CRC має важливе значення для зберігання параметрів в енергонезалежній пам'яті, оскільки випадкові стрибки напруги можуть пошкодити збережені дані, що призведе до некоректної роботи всієї системи. Тому при збереженні параметрів кожен з них, окрім свого значення, має власний циклічний надлишковий код CRC8:

```
uint8_t CRC_8(const uint8_t* pData, uint32_t Len)
{
   uint8_t CRC_8 = 0xFF;
   uint32_t i;
   while (Len--)
   {
      CRC_8 ^= *pData++;
      for (i = 0; i < 8; i++)
      CRC_8 = ( (CRC_8 & 0x80) ? (CRC_8 << 1) ^ 0x31 : CRC_8 << 1);
   }
   return CRC_8;
}
```

Для збільшення відновлення даних при їх пошкодженні, в різні ділянки енергонезалежної пам’яті записуються дві копії параметрів. Якщо при зчитуванні оригінальної частини параметрів їхні CRC не збігаються, починають зчитуватись копії. Якщо в копії всі параметри відповідають своїм циклічним залишковим кодам, тоді оригінальні дані будуть замінені на копії.

У випадку, якщо при завантаженні оригінальні та пошкоджені дані були пошкоджені, вони замінюються значеннями параметрів за замовчуванням. Ці значення ініціалізуються при включенні пристрою і постійно знаходяться в оперативній пам’яті (RAM).

Кожен пристрій має такі параметри:

1. Ідентифікаційний номер пристрою. Ідентифікаційний номер пристрою (один байт) відповідає номеру під яким пристрій буде знаходитись на шині даних. Користувач повинен враховувати, що кожен пристрій повинен мати різний ідентифікаційний номер для того щоб не відбувався конфлікт на шині даних;
2. Лічильник перезапусків системи. Цей параметр є двобайтовим і є недоступним для користувача. Він записується в пам’ять після кожного перезапуску системи і інкрементується на одиницю;
3. Період відправки паспорта пристрою на шину даних. Цей параметр (один байт) відповідає за час, через який пристрій відправлятиме свої основні дані в шину і тим самим підтверджуватиме, що він активний;
4. Версія пристрою. Це рядковий параметр, який відображає інформацію про поточну версію розробки пристрою;
5. Пароль. Пароль є рядковим параметром. Через певний інтервал часу система блокує доступ користувача до всіх параметрів. Для того, щоб отримати доступ до них, користвач має відправити пароль системі;
6. Максимальне значення напруги. Це байтовий параметр, який контролює перепади напруги. Якщо напруга стане вищою за значення цього параметру, пристрій вимкнеться, щоб унеможливити пошкодження і втрату інформації, що записана у енергонезалежній пам’яті;
7. Режим роботи пристрою. Цей параметр (один байт) визначає режим роботи пристрою. Якщо керуюча частина встановлює значення цього параметру в 0, це означає, що пристрій буде працювати в режимі виконання збірки. При значенні 1 пристрій працюватиме в режимі створення нової збірки, а при значенні 2 – в режимі паузи. Якщо значення не дорівнюватиме ні одному з вище

зазначених, сенсори пристрою ніяк не реагуватимуть на зміни своїх станів;

8. Період затримки рівня сенсора. Під час робочого процесу, працівник може випадково зачепити або доторкнутись до сенсора пристрою, який відповідає за поточний крок, а отже сигналізує про своє місцезнаходження. Параметр затримки рівня сенсора, створений для уникнення такої події. Наприклад, якщо значення цього параметру дорівнює 1, це означає, що працівник має протягом певного періоду часу доторкнутись до сенсора для переходу на наступний крок виконання роботи;
9. Лічильних неправильних кроків працівника. Коли працівник виконує неправильний крок, лічильник збільшує своє значення. Цей параметр також є системним і його неможливо змінити. Він потрібний для надання керуючій частині більшої кількості даних для детальнішого аналізу роботи працівника. Керуюча частина може обнулити цей параметр;
10. Лічильник пауз працівника. Якщо параметр режиму роботи пристрою виставлений в режим пристрою паузи, двобайтовий системний параметр буде збільшуватись на одиницю з кожним разом як користувач призупиняє виконання роботи. Цей параметр є допоміжним параметром для керуючої системи.
11. MAC адреса мінікомп’ютера Orange Pi PC Plus. Даний параметр є рядковим і створений для того, щоб виконавча і керуюча частина були прив’язані одна до одної і щоб замовник не мав змоги встановлювати власні пристрої в систему. Тому коли відбувається ініціалізація всієї системи автоматизації складських виробничих процесів, керуюча частина відправляє виконавчій свій паспорт, і якщо він не відповідатиме записаному на апаратному пристрої, система не зможе функціонувати.

Збереження і зчитування параметрів з енергонезалежної пам’яті

Організація енергонезалежної пам’яті M24P40 є секторною. Вона має вісім секторів, де кожен сектор розбитий на 256 сторінок по 256 байт. Кожна сторінка може бути окремо запрограмована, але стирання відбувається посекторно.

Робота з пам’яттю відбувається через команди, які відправляються їй у вигляді двійкового коду. Основними командами для зберігання та отримання параметрів є наступні:

1. WREN – Write Enable. Код команди – 06h. Відправка цієї команди дозволяє почати виконувати запис сторінки;
2. WRDI – Write Disable. Код команди – 04h. Відправка цієї команди забороняє запис в сторінку па'мяті;
3. READ – Read Data Bytes. Код команди – 03р. Після відправки цієї команди разом з номером сторінки, яку необхідно зчитати у відповідь приходить 256 байт інформації, зчитаної з вказаної сторінки пам’яті;
4. PP – Page Program. Код команди – 02h. Після відправки цієї команди разом з даними і номером сторінки, вказані дані будуть записані в пам’ять. Якщо об’єм даних перевищуватиме 256 байтів, дані будуть записуватись циклічно, тобто 257 байт масиву даних буде записаний в перший байт сторінки.
5. SE – Sector Erase. Код команди – D8h. Після відправки цієї команди і номеру сектора, вказаний сектор буде повністю стертий.

Параметри кожного типу записується в окрему сторінку, тобто байтові параметри записуватимуться в першу сторінку, а друга сторінка буде залишена для майбутнього розширення, двобайтові параметри записуватимуться в третю сторінку тощо. Запис відбувається побайтово. В сторінку заноситься саме значення параметра, а також його CRC. В кінці сторінки є також циклічний залишковий код, але двобайтовий (CRC16).

3.4 Система команд мікроконтролера

Для створення можливості редагування параметрів користувачем, для виконання окремих специфічних операцій в апаратному пристрої, а також для роботи з керуючою системою була розроблена система команд.

Кожний апаратний пристрій має чотирнадцять команд:

1. Getparam;
2. Setparam;
3. TPASS;
4. Bootread;
5. Bootwrite;
6. Getver;
7. Light;
8. Nextdev;
9. Dinck;
10. Booterase;
11. Bootstatus;
12. Loadparam;
13. Saveparam;
14. Rstallprof.

Пошук команд в приймальному буфері і їх виконання реалізується за модифікованим алгоритмом пошуку підрядка в рядку (рис. 3.6). Кінцевим символом кожної команди є крапка з комою. Поки алгоритм не знайде цей символ, він буде порівнювати назву кожної команди з даними, що знаходяться у прийомному буфері. Патерни команд записані у масиві:

const uint8_t* const COMMAND_STRING[]={"getparam ", "setparam ", "TPASS: ", "bootread ", "bootwrite ", "getver;", "light;", "nextdev;", "dinck;", "booterase;", "bootstatus;", "loadparam;", "saveparam;", "rstallprof;"};

Порядковий номер кожного елемента масиву, описаного вище, відповідає елементам масиву функцій команд:

typedef void(*pFunc)(RINGBUFFER*, RINGBUFFER*,uint32_t);

const pFunc Terminal_Func[]={0, GetParam, SetParam, Password, BootRead, BootWrite, GetVer, Light, Nextdev, Dinck, Booterase, Bootstatus, LoadParam, SaveParam, RstAllProf };

Завдяки цьому, після знаходження команди в буфері стає відомий її порядковий номер, тому відразу ж починається її виконання. Після виконання команди, дані з вхідного буфера, які є відповіддю на команду, стираються і продовжується пошук наступного символу крапки з комою і видалення елементів з вхідного буферу, які не задовольняють патернам команди.

Алгоритм пошуку поділяє команди на чотири категорії:

1. Команда для введення паролю;
2. Команди, які приймають параметри;
3. Команди, які не приймають параметри;
4. Команди, після яких відбувається переініціалізація системи.

Команда TPASS відноситься до першої категорії команд. Значення, яке введе користувач після задання цієї команди, порівнюється з параметром паролю, і, якщо вони збігаються, користувач на визначений час отримує доступ до всіх параметрів системи. Після того, як час з попереднього введення команди мине, доступ до системи буде заблокованим, і користувачу знову треба буде вводити пароль.

До категорії команд, які приймають параметри, відносяться чотири команди: getparam, setparam, bootwrite і bootread.

Команда getparam приймає одне значення, яке є ідентифікатором параметра. Далі відбувається перевірка існування параметра з таким ідентифікатором, і, якщо існує, визначення типу його відображення: десяткового, шістнадцяткового, або ж рядкового. Якщо такий ідентифікатор

існує, користувач у відповідь отримає значення параметра з таким ідентифікатором:

Запит: “getparam 200;”.

Відповідь: “Param ID: 200 Val: 1\r\n <200 1>\r\n”.

Якщо параметра з пошуковим ідентифікатором не існує, користувач не отримає жодної відповіді.

Команда setparam приймає два значення, розділених пробілом. Першим значенням є ідентифікатор параметра, а другим його нове значення. Якщо ідентифікатор відповідає рядковому параметру, нове значення повинне мати не більше ніж 31 символ. Якщо ідентифікатор відповідає байтовому параметру, нове значення параметра не може перебільшувати 255 тощо. Якщо ідентифікатор параметра існує і значення не відповідає його обмеженням, користувач отримає відповідь про введення некоректного значення:

Запит: “setparam 200 400;”, де ідентифікатор 200 відповідає байтовому параметру.

Відповідь: “<WRONG INPUT>”

Якщо ідентифікатор параметра існує і значення відповідає його обмеженням, вираховується циклічний надлишковий код нового значення і заноситься в таблицю параметрів. Після цього користувач отримує відповідь:

Запит: “setparam 200 400”.

Відповідь: “Param ID:0300 New Val: 10 \r\n <0300 10>\r\n”.

Якщо ідентифікатора параметра не існує в таблиці параметрів, користувач не отримає ніякої відповіді від апаратного пристрою.

Команди bootread і bootwrite виконуються, якщо пристрій має попередньо записаний завантажувач (bootloader), що зберігає основну прошивку в четвертому секторі енергонезалежної пам’яті і при включенні системи запускає її.

Завантажувач - це фрагмент коду, який дозволяє оновлювати код користувача. Нову прошивку пристрою можна отримати за допомогою альтернативних каналів завантаження, таких як USB-накопичувач. Після завантаження коду нової прошивки в ПЗУ завантажувач виконує оновлення, коли це потрібно. Спочатку за допомогою команди bootread завантажувач перевіряє фрагмети коду основної програми. Якщо нова прошивка програми відрізняється від попередньо збереженої, завантажувач стирає попередню прошивку і за допомогою команди bootwrite перезаписує сектор енергонезалежної пам’яті, в якому знаходилась попередня прошивка. Після цього він записує основну програму в пам'ять, відведену для неї. Система пристрою також виконує такі команди, як bootstatus, яка визначає, чи наявний завантажувач у пристрої, та booterase, яка очищує завантажувач.

Завантажувач розміщується за базовою адресою мікроконтролеру, завдяки чому що він почне виконуватись одразу після увімкнення пристрою (рис. 3.8).

Команди getver, light, nextdev, dinck, booterase і bootstatus відносяться до третьої категорії. Кожна з цих команд буде ідентифікована коректно при наявності в кінці символа “;”.

Команда getver відображатиме поточну версію пристрою:

Запит: “getver;”.

Відповідь: “<VER: PickToLight ver 1.21>\r\n”

Команди light, nextdev, dinck є командами, що визначають основну логіку роботи виконавчої системи. Якщо керуюча система відправляє команду light через шину даних CAN, відповідний пристрій включає світловий діод. Команда dinck вимикає світловий діод. Команда nextdev повідомляє системі, що він не повинен сигналізувати про помилку при зміні рівня на його сенсорі.

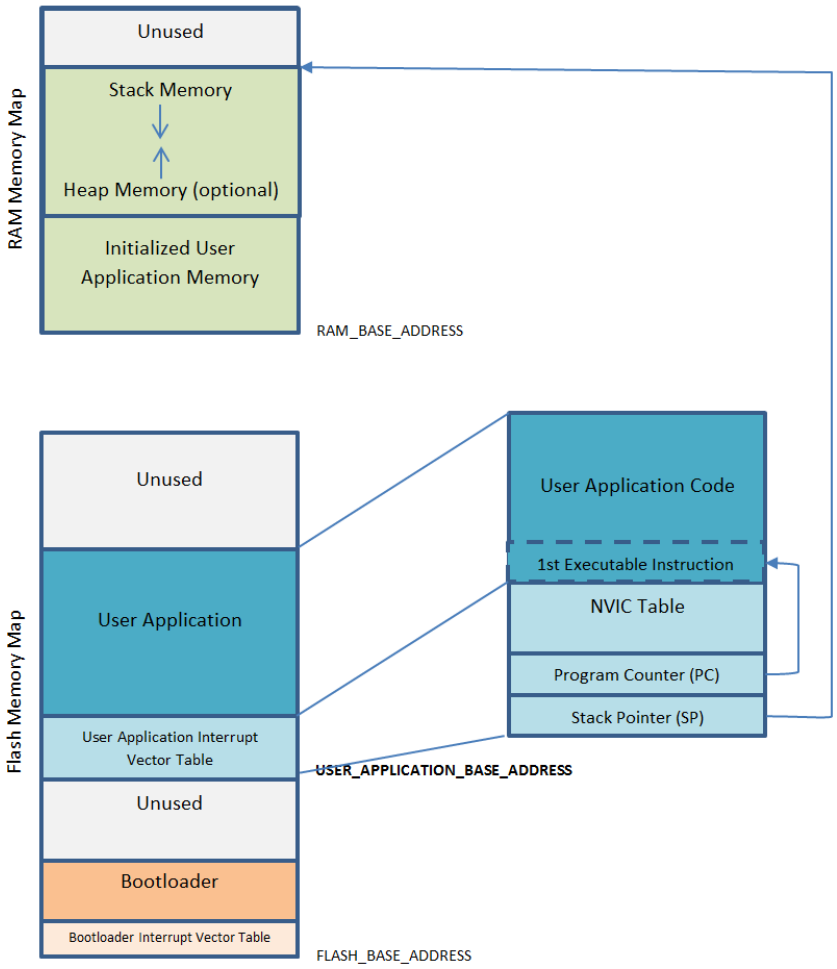

Рисунок 3.8 – Розміщення коду завантажувача та коду виконуваної програми в пам’яті мікроконтролера

До останньої категорії команд відносяться loadparam, rstallprof та saveparam. Після виконання кожної з цих команд система буде переініціалізована.

Команда saveparam виконує запис всіх параметрів в енергонезалежну пам’ять.

Команда loadparam виконує завантаження всіх параметрів з енергонезалежної пам’яті. Якщо користувач змінив параметри, і без їхнього збереження виконав цю команду, ці зміни будуть втрачені і система завантажить останню версію параметрів, що записані в енергонезалежній пам’яті.

Команда rstallprof виконує заміну всіх значень параметрів на значення за замовчуванням.

Розроблена система команд дозволяє користувачу швидко налаштувати апаратний пристрій на необхідний йому режим роботи, а також спрощує спілкування з керуючою системою засобів автоматизації складських виробничих процесів. Також вона дозволяє зберігати нові версії програмного забезпечення при наявності завантажувача.

3.5 Налаштування асинхронного приймача передавача

Асинхронний приймач UART необхідний для того, щоб користувач отримав зручний спосіб з’єднатись з пристроєм та налаштувати його. Зважаючи на значну кількість програмних засобів для роботи з UART (RealTerm [21], Termite [22] та ін.), для реалізації користувацького інтерфейсу було обрано саме цю шину даних.

Для налаштування UART необхідно визначати частоту, на якій він буде працювати, налаштувати порти введення та виведення, які відповідають за RX та TX, парність, об’єм даних кожного переданого та отриманого пакету, а також режим його роботи (режим полінгу чи режим переривань):

```
void UART_Init(void) {
    register uint32_t* Register = UART_BASE;
    register uint32_t* RegisterSYS = SYS_BASE;
    LPC_IOCON->PIO1_7 = 0xD1;
    LPC_IOCON->PIO1_6 = 0xD1;
    RegisterSYS[SYS_UARTDIV] = UART_DIV_1;
    RegisterSYS[SYS_AHBCTRL] |= UART_SET_CLK;
    Register[UART_LCR] =DLAB_ENABLE;
    Register[UART_DLM] = 0x00;
    Register[UART_DLL] = 0x17;
    Register[UART_FDR] = (0x02)|(0x0F<<4); // baudrate=115200
    Register[UART_FCR] = FIFO_ENABLE |RX_BUFFER_DEPTH_1
    Register[UART_LCR] = (((WORD_LENGTH-5)&0x03)|STOP_BIT);
    #if USE_UART_INTERRUPT
       Register[UART_IER] = IER_RBR |IER_RLS | IER_THRE;
       NVIC->ICPR[0] = 0x200000;
       NVIC->IP[UART_IRQn] = 0x80;
       NVIC->ISER[0] = 0x200000;
    #endif
}
```

У представленому вище коді шина налаштована на бітрейт 115200, восьмибітний пакет даних, непарність та на режим роботи з перериваннями. Було вирішено працювати в режимі переривань, оскільки UART є асинхронною шиною даних, тобто наперед невідомо, коли очікувати прийом даних, а також необхідність оброблення даних в прийомній черзі UART без затримки.

Коли інформація потрапляє в прийомну або вихідну чергу UART пристрій переходить до обробки його переривання:

```
void UART_IRQHandler(void) {
  uint32_t   TEMP;
  register uint32_t*  Register  = UART_BASE;
  switch(Register[UART_IIR]&(IIR_RLS|IIR_RDA|IIR_CTI|IIR_THRE)) {
     case IIR_THRE: {
                for(uint32_t i=15;i--;)
                     if(!Read_Buffer(&TX_FIFO, Register+UART_THR))
                          break;
              }
       break;
     case  IIR_CTI: {
                 if(RX_FIFO._size>RX_FIFO._mask)
                Read_Buffer(&RX_FIFO, &TEMP);
                if((TEMP = Register[UART_RBR])==';')
                    Task_Table[uart_analise].flag++;
                Write_Buffer(&RX_FIFO, TEMP);

              }
       break;
     case  IIR_RLS: {
              if (Register[UART_LSR] & (LSR_OE | LSR_BI))
                   (void)Register[UART_RBR];
              }
       break;
  default:
       break;
  }
return;
}
```

В наведеному вище коді наявні два процеси: в процесі прийому даних зчитуються отримані дані з вхідної черги та передаються в буфер даних. Після цього запускається задача, яка досліджує цей буфер даних. Якщо серед них будуть корисні дані, такі як команди, що повністю відповідають своїм паттернам, тоді відбудеться їх аналіз і обробка. Якщо корисних даних не буде знайдено, пристрій буде продовжувати свою роботу без змін.

В процесі передачі даних перевіряється вихідний буфер, і при наявності вони передаються у вихідну чергу приймача UART, який потім виводить ці дані користувачу. Такими даними можуть бути: відповіді на команди, помилка при ініціалізації інтерфейса CAN, помилка при ініціалізації портів введення та виведення, що відповідають за роботу світлових діодів та сенсорів пристрою.

Оскільки пристрій має обмежену оперативну пам’ять, необхідно реалізувати якомога компактніший спосіб зберігання вхідних та вихідних даних перед їх обробкою системою. Для реалізації цього використовується циклічний буфер.

Циклічним буфером називається структура даних, що реалізована за принципом first-in-first-out (FIFO) та має визначений розмір [23]. Такі буфери часто зустрічаються в багатьох вбудованих системах. Циклічний буфер має два вказівники до буферу. Відстань між ними може змінюватись від нуля до загальної кількості елементів у буфері. Використання двох вказівників означає, що довжина буфера може зменшуватися до нуля (у такому випадку буфер порожній) та до загальної кількості елементів (циклічний буфер заповнений). На рис. 3.9 показана структура кільцевого буфера (FIFO).

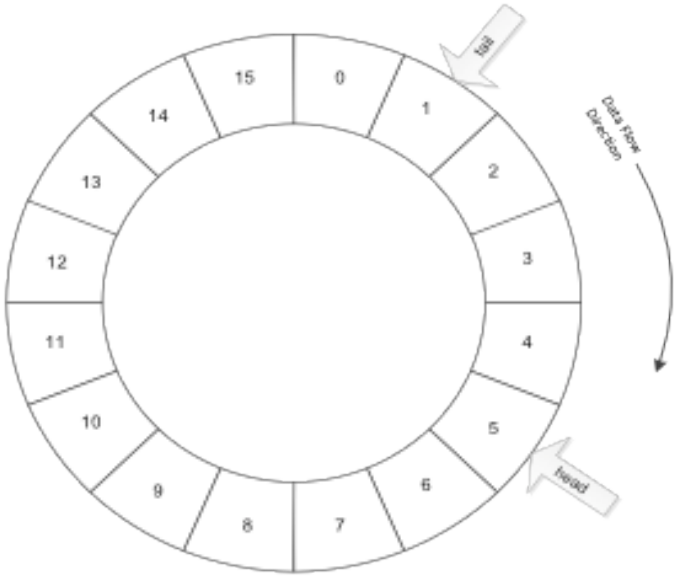

Рисунок 3.9 – Структура циклічного буфера

Принцип роботи циклічного буфера полягає у зміщенні вказівників початку та кінця буфера (рис. 3.10). Якщо буфер порожній, його вказівники знаходяться на початковій позиції буфера. Після занесення даних в буфер зміщується вказівник кінця буферу на кількість занесених даних. Після отримання даних з буфера, зміщується вказівник початку буфера на кількість отриманих даних. Якщо відстань між початком і кінцем буфера дорінює його довжині, це означає, що буфер заповнений.

Рисунок 3.10 – Принцип роботи циклічного буфера

Циклічні буфери часто використовуються як черги фіксованого розміру. Фіксований розмір є корисним для вбудованих систем, оскільки у вбудованих системах найчастіше використовуються статичні методи

зберігання даних, а не динамічні розподіли. Це зумовлено тим, що статичний метод зберігання даних в мікроконтролері зменшує ймовірність виходу коду за межі оперативної пам’яті, що призводить до збою роботи пристроя.

Циклічні буфери також є корисними структурами для ситуацій, коли внесення та отримання даних відбуваються з різною швидкістю: найновіші дані завжди доступні. Якщо отримання даних повільніше за занесення, застарілі дані будуть перезаписані останніми даними. Тому використовуючи циклічний буфер, завжди отримуємо найновіші дані.

Зважаючи на те, що працівник, який налаштовує мікроконтролер, може відправити багато команд одночасно, а також існує імовірність введення хибних даних, їх загальна кількість, яка буде передаватись через шину даних UART, може перевищувати розмір циклічного буферу. Зважаючи на це, швидкість введення та виведення даних з циклічного буферу має бути достатньою для втрати якихось даних.

Для підвищення швидкості прийому та передачі даних у циклічному буфері, вирішено реалізувати його мовою асемблеру:

```
Read_Buffer:
            PUSH {R1, R2, R3, R7, LR}
            MOVS    R3,
            CPSID   I
            LDRH    R2, [R0, #+6]
            CMP     R2, R3
            BEQ     end_read
           SUBS    R2, R2, #+1
            STRH    R2, [R0, #+6]
           LDRH    R2, [R0, #+4]
            ADDS    R2, R2, #+1
            LDRH    R7, [R0, #+10]
            ANDS    R7, R7, R2
 Read_Count
            STRH    R7, [R0, #+4]
            SUBS    R2, R2, #+1
            LDRH    R7, [R0, #+8]
            LDR     R0, [R0, #+0]
            B       buff_size
```

Мова асемблера для процесора Cortex M0 підтримує набір команд Thumb-2. Це одна з найважливіших особливостей мови, оскільки вона дозволяє спільно використовувати 16- і 32-бітні команди, забезпечуючи одночасно як високу ефективність, так і високу щільність коду. Набір Thumb-2 є гнучким, потужним і при цьому простим у використанні набором команд.

У попередніх процесорах компанії ARM центральний процесор міг знаходитися в одному з двох станів: 32-бітному ARM стані і 16-бітному Thumb стані. У стані ARM всі команди є 32-бітними, що забезпечує дуже високу продуктивність. У стані Thumb команди є 16-бітними, дозволяючи отримати більшу компактність коду. Однак ці команди мають обмежену функціональність порівняно з командами ARM, тому для виконання окремих операцій може знадобитися більша кількість команд.

Для використання переваг обох станів коду більшість прикладних програм складається з сумішей команд ARM і Thumb. Однак такий підхід не завжди себе виправдовує. На переключення процесора між станами витрачається як час, так і місце в пам'яті. До того ж наявність двох наборів команд може вимагати розбиття вихідного коду на окремі файли, кожен з яких буде компілюватись з використанням відповідного набору. Це ускладнює розробку програмного забезпечення.

З появою набору команд Thumb-2 стало можливим виконати всі необхідні операції, перебуваючи в одному стані, а необхідність переключення між двома робочими станами зникла. Тому мова асемблера процесора ARM M0 має такі переваги:

1. Відсутні накладні витрати на перемикання між станами, в результаті чого зменшується час виконання та розмір коду програми;
2. Відсутня необхідність розділення вихідних файлів на файли з кодом ARM і файли з кодом Thumb, що полегшує розробку програм та їх подальше супроводження;

3. Спрощується досягнення максимальної ефективності та продуктивності, що, в свою чергу, полегшує розробку програмного забезпечення.

3.6 Режими роботи пристроїв

Розроблювана виконавча система може працювати в двох режимах роботи: режимі створення нової збірки та режимі виконання вже існуючої збірки. Тому для того, щоб не розробляти окреме програмне забезпечення для апаратних пристроїв для кожного з режимів роботи було вирішено розробити універсальне програмне забезпечення, в якому б пристрої могли б переналаштовуватись з одного режиму в інший. Також, на кожному робочому місці має бути один апартний пристрій, який міг би призупиняти роботу системи. Таким чином кожен пристрій може бути налаштований на три різних режими.

Переключення між режимами пристрою відбувається завдяки системі параметрів пристрою. Кожний мікроконтролер має параметр режиму роботи, який може приймати чотири різні значення:

1. “0”. Кожний пристрій при першому включенні завантажує в параметр саме це значення, оскільки воно встановлене за замовчуванням. Якщо параметр режиму роботи має це значення, пристрій ніяк не реагуватиме на зміни сенсору мікроконтролера і команди, які йому відправлятиме керуюча частина;
2. “1”. Якщо працівник при налаштуванні пристрою встановлює параметр режима роботи значення “1”, це означає, що він переходить до режиму виконання вже існуючої збірки. Тобто, що при виконанні збірки на етапі, якому відповідатиме адреса пристрою, прийде команда “light” і його індикатор почне світити зеленим кольором. Після того, як працівник виконає цей етап і доторкнеться до сенсору

на мікроконтролері, світло погасне і засвітиться світловий діод пристрою, що відповідає наступному кроку виконання збірки;

3. “2”. Якщо працівник при налаштуванні мікроконтролера встановлює значення “2”, мікроконтролер почне працювати в режимі створення нової збірки. У такому випадку, алгоритм роботи пристрою полягатиме в тому, що, при введенні ідентифікатора пристрою керуючою системою, пристрій отримає команду “light” і протягом певного інтервалу часу миготінням індикатора повідомлятиме про своє місцезнаходження працівнику;
4. “3”. Якщо працівник при налаштуванні мікроконтролера встановлює значення “3”, пристрій почне працювати у режимі призупинення робочого процесу. Якщо під час виконання збірки працівник доторкнеться до сенсору, налаштованого на цей параметр мікроконтролера, його індикатор почне світити синім кольором, а керуючій системі засобів автоматизації складського виробничого процесу буде відправлене повідомлення про необхідність призупинення виконання. Система запам’ятає поточний крок виконання і запише в параметр режиму роботу кожного пристрою, що брав участь у виконанні збірки, значення “0”. Після повторного дотику до пристрою, індикатор погасне. Керуюча система перевірить, чи всі пристрої все ще доступні, і налаштує їх знову на режим виконання збірки, а також відновить крок робочого процесу, на якому зупинився працівник.

Висновки до розділу 3

Розглянуто взаємодію керуючої та виконавчої систем через протокол зв’язку CAN. Оскільки виконавча система виконує багато як основних, так і допоміжних функцій, що задовольняють основним вимогам системи автоматизації, а також повинна працювати у режимі реального часу, було розроблено операційну систему реального часу.

При розробці виконавчої системи запропоновано декілька технічних рішень, які дозволяють забезпечувати завадостійкість системи, її динамічну реконфігурацію та прискорювати її роботу для зручнішої взаємодії з користувачем.

Зважаючи на те, що апаратні пристрої можуть пошкодитись, розроблено систему параметрів, яка дозволяє зробити налаштування кожного пристрою гнучким, що надає можливість заміни пошкоджених пристроїв.

Для комунікації виконавчої та керуючої систем була розроблена система команд, яка дозволяє виконавчій системі налаштовуватись на відповідні режими роботи, виконувати дії з індикаторами, а також надавати необхідні для роботи керуючої системи дані.

Для налаштування пристроїв використовується асинхронний приймач передавач UART. Для роботи з UART запропоновано одне з ефективних рішень для пристроїв з малою оперативною пам’яттю - використання циклічного буфера. Для прискорення роботи з буфером (отримання та відправлення даних з нього) відповідний код реалізовано мовою асемблера процесора Cortex ARM M0.

4. РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ ВИКОНАВЧОЇ СИСТЕМИ

4.1 Результати аналізу досліджень виконавчої системи

Процесори мікроконтролерів розробленої виконавчої системи засобів автоматизації складських виробничих процесів мають обмежений об’єм пам’яті, а саме 8 Кбайт оперативної пам’яті та 24 Кбайт постійної пам’яті. Тому виникає потреба ОСРЧ, об’єм якої буде невеликим.

Виконавча система було портована на відомі ОСРЧ FreeRTOS та ChibiOS, а також на розроблену ОС, яка виконує лише той функціонал, що необхідний для реалізації засобів автоматизації. Результати, наведені в таблиці 4.1 показали, що кожна ОСРЧ задовольняє обмеженням пам’яті процесора, проте для майбутнього розширення ресурсів пам’яті, які залишились може буде недостатньо.

Таблиця 4.1 – Порівняння об’єму використаної пам’яті розглянутими операційними системами реального часу

Тип пам’яті	FreeRTOS	ChibiOS	Розроблена ОС
ROM	6831 байт	6425 байт	4321 байт
RAM	3871 байт	3735 байт	2352 байт

Реалізація взаємозв’язку між виконавчою та керуючою системами, а також між виконавчою системою та користувачем відбувається відбувається протоколами зв’язку, які використовують циклічний буфер, який відповідає за введення та виведення даних.

Циклічний буфер був реалізований двома мовами програмування: мовою С та мовою асемблера. Проведені досліди показали, що введення і виведення даних відбувається швидке, якщо циклічний буфер написаний мовою асемблера (таб. 4.2). Досліди були проведені, коли єдиним процесом

ОСРЧ була робота з циклічним буфером. Зважаючи на це, при реалізації всіх процесів ОСРЧ виконавчої системи, час введення та виведення даних через протоколи зв’язку збільшиться.

Таблиця 4.2 – Порівняння роботи цикличного буферу, написаного мовами программя С та асемблера

Розмір переданих даних	Зчитування з буфера даних мовою С	Відправка в буфер даних мовою С	Зчитування даних мовою асемблера	Відправка даних мовою асемблера
10 Кбайт	1418 мкс	1243 мкс	843 мкс	734 мкс

Висновки до розділу 4

Розроблена виконавча система засобів автоматизації складських процесів була виконана та досліджена на трьох операційних системах реального часу. Аналіз обсягів використаної пам’яті операційних систем показав, що розроблена операційна система реального часу є компактнішою, що дозволить розширювати функціональність засобів автоматизації в подальшому.

Результати аналіз роботи циклічного буфера, написаного мовами С та асемблера показують, що ефективніше використовувати циклічний буфер, написаний мовою асемблера, оскільки це пришвидшить взаємодію апаратного пристрою та користувача.

ВИСНОВОК

Для багатьох компаній актуальною проблемою є забезпечення ефективного налагодження та прискорення виконання виробничого циклу. Суттєве місце для її вирішення відводиться засобам автоматизації складських процесів.

Зважаючи на це, досліджено декілька відомих підходів для автоматизації складських виробничих процесів, таких як Pick to light, Pick by voice, метод радіочастотної ідентифікації, Augmented reality picking тощо. На основі аналізу їхніх недоліків та переваг вирішено використати для розробки засобів автоматизації принцип Pick to light з модифікаціями, які дозволяють усунути окремі недоліки цього підходу.

При проектуванні засобів автоматизації, їх було вирішено поділити на дві, взаємодіючі одна з одною підсистеми: керуючу та виконавчу, яка і розглядалась в магістерській дисертації. Виконавча система є сукупністю апаратних пристроїв (мікроконтролерів), кожен з яких працює на ядрі Cortex ARM M0, який забезпечує необхідний функціонал для виконання вимог до засобів автоматизації. Обґрунтовано вибір протоколів шин даних, які забезпечують комунікацію між виконавчою та керуючою системами, між мікроконтролером та працівником, а також між мікроконтролером та енергонезалежною пам’яттю, яка необхідна для динамічного налаштування пристрою.

При розроблюванні програмного забезпечення використовувалась мова програмування C та мова асемблера для процесора ARM Cortex M0. Програмне забезпечення було скомпільоване в середовищі розробці IAR Embedded Workbench.

Для мікроконтролерів розроблено операційну систему реального часу. Запропонована оптимізація параметрів системи дозволяє зменшити обсяг пам’яті для збереження її коду, що вагомо для процесорів цієї серії, а також в подальшому дозволить розширювати функціональність засобів.

Розроблена система параметрів пристрою дозволяє швидко налаштовувати мікроконтролер на роботу в різних режимах. Створена система команд реалізовує протокол зв’язку між керуючою та виконавчими системами. Для прискорення обробки, введення та виведення даних окремі фрагменти коду реалізовані мовою асемблера для процесора Cortex ARM M0.

Результати тестових випробувань засобів автоматизації, зокрема виконавчої системи, доводять їх функціональність, відповідно до поставлених вимог, ефективність та оптимальність витрат на реалізацію.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Кіндзер М.С., Курилич Р.А., Дробязко І.П. Комп'ютерний моніторинг і автоматизація складських виробничих процесів Прикладна математика та комп'ютинг. ПМК, 2018: зб. тез доп. – К.: Просвіта, 2018 – 28-33 с.
2. Таран С.А. Як організувати склад – М.: "Альфа-Пресс", 2008. – 240 с.
3. WMS System – Електон. дані (1 файл) – Режим доступу: https://searcherp.techtarget.com/definition/warehouse-management-system-WMS/ (дата звернення 17.09.2018) – Назва з екрану.
4. Enterprise Resource Planning and Warehouse Management System – Електон. дані (1 файл) – Режим доступу: http://www.qstockinventory.com/blog/warehouse-management-system-vs-enterprise-resource-planning/ (дата звернення 19.09.2018) – Назва з екрану.
5. The 8 Best Order Picking Methods Електон. дані (1 файл) – Режим доступу: https://www.newcastlesys.com/blog/bid/348476/order-picking-methods-and-the-simplest-ways-to-minimize-walking-infographic (дата звернення 22.09.2018) – Назва з екрану.
6. Order picking basics – Електон. дані (1 файл) – Режим доступу: https://www.logisticsmgmt.com/article/order_picking_basics (дата звернення 25.09.2018) – Назва з екрану.
7. How To Design Your Own Pick List – Електон. дані (1 файл) – Режим доступу: https://www.logiwa.com/blog/design-your-own-pick-list (дата звернення 26.09.2018) – Назва з екрану.
8. What is Pick to light? – Електон. дані (1 файл) – Режим доступу: https://www.bastiansolutions.com/solutions/technology/supply-chain-software/picking-technology/pick-to-light/ (дата звернення 06.10.2018) – Назва з екрану.

9. Radio Frequency Identification – Електон. дані (1 файл) – Режим доступу: https://www.datexcorp.com/hardware/rfid/ (дата звернення 10.10.2018) – Назва з екрану.
10. Pick by voice – Електон. дані (1 файл) – Режим доступу: https://www.optiscangroup.com/solutions/warehouse-solutions/pick-by-voice (дата звернення 13.10.2018) – Назва з екрану.
11. A smart order picking system supported by augmented reality – Електон. дані (1 файл) – Режим доступу: https://www.tsystems.com/de/en/references/use-cases/use-case/pick-by-vision-633422 (дата звернення 15.10.2018) – Назва з екрану.
12. Джозеф Ю. Ядро Cortex-M0 компании ARM. Полное руководство – М.: "ДМК", 2015. – 10 с., 45-68 с.
13. Basics of UART Communication – Електон. дані (1 файл) – Режим доступу: https://www.techopedia.com/definition/3669/universal-asynchronous-receivertransmitter-uart (дата звернення 21.10.2018) – Назва з екрану
14. Basics of the SPI communication protocol – Електон. дані (1 файл) – Режим доступу: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/ (дата звернення 25.10.2018) – Назва з екрану
15. CAN bus explained – Електон. дані (1 файл) – Режим доступу: https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en/ (дата звернення 25.10.2018) – Назва з екрану
16. MP25P40 datasheet – Електон. дані (1 файл) – Режим доступу: https://www.digchip.com/datasheets/parts/datasheet/3454/M25P40-VMN3PBA.php (дата звернення 25.10.2018) – Назва з екрану
17. Basics of Embedded C Program – Режим доступу: https://www.electronicshub.org/basics-of-embedded-c-program/ (дата звернення 26.10.2018) – Назва з екрану

18. GCC and IAR compiler comparison – Режим доступу: http://nxtgcc.sourceforge.net/wiki/GCC_and_IAR_compiler_comparison (дата звернення 30.10.2018) – Назва з екрану
19. ОС реального времени FreeRTOS – Електон. дані (1 файл) – Режим доступу: http://rus-linux.net/MyLDP/BOOKS/Architecture-Open-Source-Applications/Vol-2/freertos-04.html (дата звернення 04.11.2018) – Назва з екрану
20. Cyclic Redundancy Check (CRC) – Електон. дані (1 файл) – Режим доступу: https://www.techopedia.com/definition/1793/cyclic-redundancy-check-crc (дата звернення 05.11.2018) – Назва з екрану
21. RealTerm – Електон. дані (1 файл) – Режим доступу: https://sourceforge.net/projects/realterm/ (дата звернення 05.11.2018) – Назва з екрану
22. Termite – Електон. дані (1 файл) – Режим доступу: https://www.compuphase.com/software_termite (дата звернення 05.11.2018) – Назва з екрану
23. Circular buffer – Електон. дані (1 файл) – Режим доступу: https://embeddedartistry.com/blog/2017/4/6/circular-buffers-in-cc (дата звернення 19.11.2018) – Назва з екрану